JOURNAL OF COMPUTER AND SYSTEM SCIENCES 40, 229–268 (1990)

Parametrization for Order-Sorted Algebraic Specification

AXEL POIGNÉ*

GMD F2G2, Schloss Birlinghoven, Postfach 1240, D-5205 St. Augustin, West Germany

Received July 28, 1987; revised December 3, 1987

We investigate parametrization for order-sorted algebraic specifications. As a prerequisite we study free constructions for order-sorted algebras and relate the various approaches to order-sorting. Then we analyse parameter passing, the result being that the notion of order-sorted specification has to be restricted in order to establish our main result, namely, that parameter passing satisfies the same correctness criteria as in the case of many-sorted algebras.

1. INTRODUCTION

Standard many-sorted algebra provides a rather restricted notion of algebra. Conditional equations dramatically increase the power of description without yet fully exploiting the concept of "algebraicity".[1] A less radical deviation from standard algebra introduces subsorts and overloading of operators.

EXAMPLE.

pres NUMBER **is**
sorts $nat < int$
ops $0: \to nat$
$\mathrm{suc}, \_-1: nat \to nat$
$\_+\_: nat\ nat \to nat$
$\mathrm{suc}, \mathrm{pred}: int \to int$
$\_+\_: int\ int \to int$
var $x, y: int, z: nat$
eqns $\mathrm{pred}(\mathrm{suc}(x)) = x$
$\mathrm{suc}(\mathrm{pred}(x)) = x$
$\mathrm{pred}(x) + y = \mathrm{pred}(x + y)$
$\mathrm{suc}(x) + y = \mathrm{suc}(x + y)$
$0 + y = y$
$0 - 1 = 0$
$\mathrm{suc}(z) - 1 = z$

* The text was prepared while at Dep. of Computing, Imperial College, London.

[1] The border line of algebra being marked by theories being *essentially algebraic* [10, 5], *generalized algebraic* [27], *equational partial* [2], which are equivalent [25], all being an instance of *locally presentable categories* [12].

0022-0000/90 $3.00

The notation is quite suggestive. One has, however, to be careful that overloaded operators behave in the same way on the same data, e.g., suc: *nat* → *nat* like suc: *int* → *int* on data of sort *nat*.

The example exposes two main features of order-sorted algebras:

- Operations can be defined partially, e.g., the -1 is only defined on the natural numbers.
- The scope of equations can be restricted, e.g., $\mathrm{suc}(z) - 1 = z$ only holds for data of type *nat*.

Both features become even more evident in the following specification of the inevitable stack[2]

spec NUMBERSTACK **is**
NUMBER **with**
sorts *nestack* < *stack*
ops empty: → *stack*
push: *stack int* → *nestack*
pop: *nestack* → *stack*
top: *nestack* → *int*
var *s*: *stack*, *m*: *int*
eqns pop(push(*s*, *m*)) = *s*
top(push(*s*, *m*)) = *m*

which avoids the problem of attaching meaning to "top(empty)".

Order-sorting is less expressive than conditional equations but it allows for a less baroque and more lucid style of specification for certain classes of algebras otherwise specified by conditional equations. One can, moreover, expect a more convenient representation of standard constructions which facilitates the task of arguing about such algebras. The second section discusses the representation of free constructions. Moreover, the various approaches to be found in the literature will be related.

Parametrization is acknowledged as one of the major techniques used for modularization of software systems. Any specification method will and should be judged for the parametrization facilities provided. We investigate parametrization for order-sorted specifications in the third section. Behaviour of parametrization substantially depends on the definition of parametrized specifications as order-sorting affects correctness of parameter passing. In order to retain the result of [8] we consider order-sorted specifications where every operator has a "maximal" instance. Moreover, parametrized specifications have strictly to separate between formal parameter and body in that no overloading of operators takes place.

This article is an expanded and improved version of [23, 24]. We assume familiarity with [7] though the paper is self-contained.

[2] To anticipate a common misunderstanding: stacks are more often used to point out the deficiencies of algebraic specification.

2. Order-Sorted Algebras and Theories

2.1. *Definitions*

Order-sorted algebras are introduced in [15]. Gogolla [13] (see also [14]) provides another formalization while [19] gives a more precise account of [15]. Analysis proves that these approaches have a different account of overloading. Our approach follows the lines of [13].

All authors agree that

- the partial ordering on sorts expresses a subset relation on the carrier sets of an algebra and that
- overloading of operators is used in that several types may be attached to operator names.

Definition. An (*order-sorted*) *signature* SIG consists of

- a partially ordered set $S = (S, \leqslant)$ of sorts, and
- an indexed set of operators $\Sigma = (\Sigma_{w,s} \mid w \in S^*, s \in S)$

(which are not necessarily disjoint).

The set of *operator names* is $\Sigma = \bigcup_{w \in S^*, s \in S} \Sigma_{w,s}$.

While little dispute arises about the meaning of the specifications considered in the Introduction, the following causes some disagreement

Example.

pres NUMBER + BOOL **is**
NUMBER **with**
sorts *bool*
ops true, false: $\rightarrow$ *bool*
_+_: *bool bool* $\rightarrow$ *bool*
eqns true + true = true
true + false = true
false + true = true
false + false = false

A. Operators may be distinguished only if their names (excluding the type information) are different. Operations applied to the same data should yield the same results independently of typing. The following definition which we will adhere to in this paper encodes this view

Definition A. An (order-sorted) SIG-*algebra* A consists of

- an S-indexed set $A = (A_s \mid s \in S)$, and
- a function $\sigma_A^{w,s}: A_w \rightarrow A_s$ for each $\sigma: w \rightarrow s \in \Sigma$ s.t.
  (i) $A_s \subseteq A_{s'}$ if $s \leqslant s'$, and
  (ii) $\sigma_A^{w,s}(\mathbf{a}) = \sigma_A^{w',s'}(\mathbf{a})$ if $\sigma: w \rightarrow s$, $\sigma: w' \rightarrow s' \in \Sigma$, $\mathbf{a}: w$, $\mathbf{a}: w' \in A$.

A *homomorphism* $h: A \to B$ is an S-indexed function such that

(i) $h_s(\sigma_A^{w,s}(\mathbf{a})) = \sigma_B^{w,s}(h_w(\mathbf{a}))$ for $\sigma: w \to s \in \Sigma$ and $\mathbf{a}: w \in A$

(ii) $h_s(a) = h_{s'}(a)$ if $a: s$, $a: s' \in A$.

The category of SIG-algebras and SIG-homomorphisms is denoted by $\mathbf{OSAlg}_{\mathrm{SIG}}$.

Notation. We will use an ambiguous but convenient notation for indexed sets in that the same name refers to the indexed set as to the union of its components, i.e., $A = \bigcup_{s \in S} A_s$. The meaning is in general obvious from the context. We often use $a \in A$ to state that $\mathbf{a}$ is in the union while $a: s \in A$ is a programming style notation for $a \in A_s$. $\mathbf{a}: w \in A$ stands for $\mathbf{a} \in A_w$, where $A_\lambda = \{\varnothing\}$ and $A_{ws} = A_w \times A_s$, S-sorted functions, relations, etc. are canonically extended to tuples.

Remarks. • Our definition differs from that of [13] as the condition $\Sigma_{w,s} \subseteq \Sigma_{w',s'}$ if $s \leqslant s'$ and $w' \leqslant w$ has been dropped which does not affect the category of models (up to isomorphism).

• One may alternatively use operator sets with multivalued sort mappings sort: $\Sigma \to S^+$ and multivalued mappings $\mathrm{sort}_A: A \to S$ to express the overloading. Then the algebra operations become partial mappings with suitable additional conditions.

• The definition ensures that the mappings

$$\sigma_A: \bigcup_{i \in I} A_{w_i} \to \bigcup_{i \in I} A_{s_i}, \qquad \sigma_A(\mathbf{a}) := \sigma_A^{w_i, s_i}(\mathbf{a}) \qquad \text{if} \quad \mathbf{a}: w_i \in A$$

are well defined for operator names $\sigma \in \Sigma$, where $\sigma: w_i \to s_i \in \Sigma$, $i \in I$, are all the typed instances of σ. σ_A may be thought of as the *meaning* of σ.

Similarly, each homomorphism defines a mapping $h: A \to B$ on the unions of the carriers.

• Standard many-sorted algebra [7] is subsumed if we assume disjointness of the operator sets.

B. Alternatively, an "obvious" procedure is to disambiguate addition and disjunction by adding type information to the operator names. This is the view taken in [19]. Overloaded operators are to be disambiguated by the type information except if they are related by *coercion*. I.e., $\sigma: w \to s \leqslant \sigma: w' \to s'$ if $w \leqslant w'$ and $s \leqslant s'$ (the ordering being canonically extended to sort words). Semantically:

DEFINITION B. An (order-sorted) SIG-*algebra* consists of a carrier set A_u, a family $(A_s \mid s \in S)$ of subsets of A_u, and a function $\sigma_A^{w,s}: A_w \to A_s$ for each $\sigma: w \to s \in \Sigma$ s.t.

(i) $A_s \subseteq A_{s'}$ if $s \leqslant s'$, and

(ii) $\sigma_A^{w,s}(\mathbf{a}) = \sigma_A^{w',s'}(\mathbf{a})$ if $\sigma: w \to s \leqslant \sigma: w' \to s'$, $\mathbf{a}: w \in A$.

(One may choose to include u as a maximal sort in S.)

With homomorphisms being functions $h: A_u \to B_u$ such that $h(\sigma_A^{w,s}(\mathbf{a})) = \sigma_B^{w,s}(h(\mathbf{a}))$ for $\sigma: w \to s \in \Sigma$ and $\mathbf{a}: w \in A$ this defines a category $\mathbf{OS_B Alg B}_{\mathrm{SIG}}$ for every signature SIG.

A trivial example demonstrates the difference of the definitions A and B: Let TRIV $= (s, s', a: \to s, a: \to s')$, and let $\{b, c\}$ be a carrier set for sort s and for sort s'. Definition B allows us to disambiguate the operators in that one can interpret the operator $a: \to s$ by b and $a: \to s'$ by c, since there is no coercion. Standard terminology refers to "ad hoc polymorphism." Definition A restricts the interpretation in that both $\mathbf{a}: \to s$ and $a: \to s'$ must be interpreted either by b or by c.

Goguen and Meseguer [19] favour ad hoc polymorphism. In contrast, I believe it to be a virtue of Definition A to exclude ad hoc polymorphism which, as I claim, does not enhance lucidity of specifications. Hence I would rather reject presentations such as NUMBER + BOOL because of poor style but suggest using different names for operators which substantially differ in meaning.[3] The definition of Goguen and Meseguer is motivated by the observation:

If we weaken the subset relations $A_s \subseteq A_{s'}$ to injective functions $m: A_s \to A_{s'}$, this does not essentially change the nature of order-sorted algebras in that one obtains an equivalent category of many-sorted algebras, some provisos being satisfied. More precisely, subsorting $s \leqslant s'$ is replaced by *monomorphic* operators $m_{s,s'}: s \to s'$ such that

(i) $x = y$ if $m_{s,s'}(x) = m_{s,s'}(y)$ and

(ii) $m_{s',s''}(m_{s,s'}(x)) = m_{s,s''}(x)$ and $m_{s,s}(x) = x$.

Coercion of operators can be expressed by adding equations

$$\sigma^{w',s'}(m_{s_1,s_1'}(x_1), \ldots, m_{s_n,s_n'}(x_n)) = m_{s,s'}(\sigma^{w,s}(x_1, \ldots, x_n))$$

where $w = s_1 \cdots s_n$, $w' = s_1' \cdots s_n'$ with $s_i \leqslant s_i'$ and $s \leqslant s'$.

Let CESIG be the specification obtained by enriching SIG in the way suggested. Order-sorting then is reduced to specification with conditional equations the semantics of which is well understood:

2.1. PROPOSITION (Goguen and Meseguer [19]). $\mathbf{OS_B Alg}_{\mathrm{SIG}}$ *and* $\mathbf{Alg}_{\mathrm{CESIG}}$ *are equivalent.*

Sketch of the Proof. (Our proof is slightly different from that in [19] as we assume that usually carrier sets are disjoint.) Given a $\mathbf{OS_B Alg}_{\mathrm{SIG}}$-algebra A we construct a CESIG-algebra B with carriers $B_s = \{(s, a) \mid a \in A_s\}$, operators $\sigma_B^{w,s}((s_1, a_1), \ldots, (s_n, a_n)) = (s, \sigma_A^{w,s}(a_1, \ldots, a_n))$ and monomorphisms $m_{s,s'}((s, a)) = (s', a)$. Vice versa, given a CESIG-algebra B we construct a $\mathbf{OS_B Alg}_{\mathrm{SIG}}$-algebra A

[3] Added in proof: The conflict between versions A and B seems to be resolved since Goguen and Meseguer opt for version A in [19].

by $A_s = \{m_s(b) \in B \mid b \in B_s\}$, where $B = (\amalg_{s \in S} B_s)_{/\approx}$ with $b \approx b'$ iff $\exists s'' \in S: m_{s,s''}(b) = m_{s',s''}(b')$ for $b \in B_s$, $b' \in B_{s'}$ (in other words the colimit of the obvious diagram, $\amalg$ denotes the disjoint union) and where $m_s: B_s \to B$ are the canonical injections (!). Operations are given by

$$\sigma_A^{w,s}(m_{s_1}(b_1), ..., m_{s_n}(b_n)) = m_s(\sigma_B^{w,s}(b_1, ..., b_n)).$$

This determines the object parts of the equivalence which is rather cumbersome to check. ∎

At a superficial inspection the order-sorted algebras of type A do not have an equivalent specification using conditional equations. We observe, however, that implicitly "intersections" have been used. These can be added formally in that we construct the free lower semilattice $S^\sqcap$ over the partially ordered set S of sorts.[4] Explicitly, $S^\sqcap$ is determined by the preorder $X \sqsubseteq Y :\Leftrightarrow \forall y \in Y \exists x \in X: x \leqslant y$ on the power set of S (factorization by antisymmetry yields $S^\sqcap$). We then add operators:

$$\sigma: \bigsqcap_{j \in J} w_j \to \bigsqcap_{j \in J} s_j \qquad \text{for} \quad J \subseteq I,$$

where $\{\sigma: w_i \to s_i \mid i \in I\}$ is the set of all operators with name σ in SIG (the meet is defined on components for words). Let $\mathrm{SIG}^\sqcap$ be the resultant signature.

2.2. PROPOSITION. $\mathbf{OSAlg}_{\mathrm{SIG}}$ *and* $\mathbf{OS_B Alg}_{\mathrm{SIG}^\sqcap}$ *are equivalent provided that SIG has a maximal sort.*

Idea of the Proof. One way is obvious in that one interprets the "intersection sorts" by the respective intersections. On the other hand, one uses exactly the same colimit construction as above but only considers the sorts by SIG. Again the details are cumbersome. ∎

On the contrary, we can disambiguate every signature in that we factorize the disjoint union $\amalg_{w \in S^*, s \in S} \Sigma_{w,s}$ by the least equivalence containing coercion. Let DSIG be the resultant signature.

2.3. PROPOSITION. $\mathbf{OS_B Alg}_{\mathrm{SIG}}$ *is isomorphic to* $\mathbf{OS_B Alg}_{\mathrm{DSIG}}$ *is isomorphic to* $\mathbf{OSAlg}_{\mathrm{SIG}}$.

Idea of the Proof. The disambiguation is an isomorphism of signatures which does not affect the semantics as the restrictions only apply to operators which are in the equivalence. The change of the concepts of algebras does not affect the semantics as only the operators in the equivalence are overloaded. ∎

Remark. We refrain from illustrating the constructions by an example as the constructions are straightforward and as even simple examples take a considerable amount of space. However, we suggest that the reader might work out the details for the signature $\mathrm{SIG} = \{s < s' < u,\ s < s'' < u,\ a: \to s,\ a: \to s''\}$.

[4] The lower semilattice $S^\sqcap$ has been introduced in [16]. I have not been aware of the reference.

These observations suggest that the approach of [Goguen-Meseguer 85] and the one advocated here differ only conceptually. In fact, we will argue further below that the distinctions are marginal if "term generated" algebras are considered. Hence preference can be given only on conceptual and pragmatic grounds.

2.2. *Constructing Order-Sorted Algebras*

All authors depend on the

DEFINITION. SIG $= (S, \Sigma)$-*terms* over an S-indexed set $X = (X_s \mid s \in S)$ of variables are inductively defined by

(i) $x: s \in T_{\mathrm{SIG}}(X)$ if $x: s \in X$

(ii) $\sigma(\mathbf{t}): s' \in T_{\mathrm{SIG}}(X)$ if $\sigma: w \to s \in \Sigma$, $\mathbf{t}: w \in T_{\mathrm{SIG}}(X)$ and $s \leqslant s'$

(we assume $\Sigma \cap X = \varnothing$ without restriction of generality).

With operations

$$\sigma^{w,s}: T_{\mathrm{SIG}}(X)_w \to T_{\mathrm{SIG}}(X)_s, \qquad \mathbf{t} \to \sigma(\mathbf{t})$$

this defines a SIG-algebra $T_{\mathrm{SIG}}(X)$.

We will see below that $T_{\mathrm{SIG}}(X)$ is "free" over X provided that X is a "order-sorted set." This does not hold for order-sorted algebras of type B [19]:

Given a signature TRIV $= (s, s', a: \to s, a: \to s')$ there exists no homomorphism to the TRIV-algebra given by $A = \{b, c\}$, $A_s = \{b\}$, $A_{s'} = \{c\}$ with obvious operations. Thus Goguen and Meseguer assume *regularity* of signatures, i.e., for any $w'' \in S^*$ s.t. there is a $\sigma: w \to s \in \Sigma$ with $w'' \leqslant w$ then there exists a least (w.r.t. coercion) $\sigma: w' \to s' \in \Sigma$ s.t. $w'' \leqslant w'$. Regularity guarantees "freeness" of the term construction provided that the sets X_s are disjoint. We just observe that our notion of algebra implicitly states regularity in that all necessary operators on "intersections" exist "virtually" (which has been used in the previous section).

In order to exploit the implicit structure of SIG-algebras we introduce the subset structure as well for variables.

DEFINITION. Given a partially ordered set $S = (S, \leqslant)$ an *order-sorted* (w.r.t. S) *set* X is an S-indexed family $X = (X_s \mid s \in S)$ of sets such that $X_s \subseteq X_{s'}$ if $s \leqslant s'$. A *morphism* $f: X \to Y$ of order-sorted sets is an S-indexed family $(f_s: X_s \to Y_s \mid s \in S)$ of functions such that $f_s(x) = f_{s'}(x)$ if $x: s$, $x: s' \in X$. The category of order-sorted sets (w.r.t. S) is denoted by $\mathbf{Set}^S$. There is a forgetful functor $V: \mathbf{OSAlg}_{\mathrm{SIG}} \to \mathbf{Set}^S$ of SIG-algebras to the underlying order-sorted sets.

Notation. Order-sorted sets may be specified by listing the elements by $\{x_i: s_i \mid i \in I\}$ denoting the order-sorted set $\{x_i: s_i \mid i \in I, s_i \leqslant s\} \mid s \in S\}$.

Before we set our to prove that the term construction provides a free algebra, a remark about the induction principle to be used may be appropriate:

Induction is on terms $T_{\mathrm{SIG}}(X)$ (here the union of the indexed set) as expressed by the second-order statement

$$\forall P: \left[\forall x \in X: P(x) \wedge \bigwedge_{\sigma: w \to s \in \Sigma} \forall \mathbf{t} \in T_{\mathrm{SIG}}(X): P(\mathbf{t}) \wedge \sigma(\mathbf{t}) \in T_{\mathrm{SIG}}(X) \to P(\sigma(\mathbf{t}))\right] \to \forall t \in T_{\mathrm{SIG}}(X): P(t).$$

Predicates P are typically specified by indexed predicates P_s, $s \in S$. P then is defined by

$$P(t) = \left[\bigvee_{s \in S} t: s \in T_{\mathrm{SIG}}(X) \wedge P_s(t)\right] \wedge \bigwedge_{s,s' \in S} [t: s \in T_{\mathrm{SIG}}(X) \wedge t: s' \in T_{\mathrm{SIG}}(X) \to [P_s(t) \leftrightarrow P_{s'}(t)]].$$

The conditions ensure that the indexed predicates only depend on the structure but not on the type information of terms. In inductions we, in general, distinguish two cases: $t = x \in X$ and $t = \sigma(\mathbf{t})$. There always exists a sort s' such that $t: s' \in T_{\mathrm{SIG}}(X)$ by construction; specifically, if $t = \sigma(\mathbf{t})$ there exists a $\sigma: w \to s \leqslant s'$ s.t. $\mathbf{t}: w \in T_{\mathrm{SIG}}(X)$.

2.4. PROPOSITION. *$T_{\mathrm{SIG}}(X)$ is a free SIG-algebra over the order-sorted set X w.r.t. V: $\mathbf{OSAlg}_{\mathrm{SIG}} \to \mathbf{Set}^S$.*

Proof. Let $t: s \in T_{\mathrm{SIG}}(X)$ and $s \leqslant s'$.

(a) $t = x: s \in X$. Then $x: s' \in X \subseteq T_{\mathrm{SIG}}(X)$ as X is a order-sorted set.

(b) $t = \sigma(\mathbf{t})$. Then there exist $\sigma: w \to s'' \leqslant s$ and $\mathbf{t}: w \in T_{\mathrm{SIG}}(X)$. As $s'' \leqslant s$ we have $\sigma(\mathbf{t}): s' \in T_{\mathrm{SIG}}(X)$. This proves condition (i) of SIG-algebras. For (ii) let $\sigma: w \to s$, $\sigma: w' \to s'$, and $a: w$, $a: w' \in T_{\mathrm{SIG}}(X)$. Then $\sigma^{w,s}(\mathbf{a}) = \sigma(\mathbf{a}) = \sigma^{w',s'}(\mathbf{a})$.

The unit is $X\eta: X \to T_{\mathrm{SIG}}(X)$, $x \to x$. Now assume that A is a SIG-algebra and that $f: X \to V(A) \in \mathbf{Set}^S$. We define

$$f_s^{\#}(x) = f_s(x) \qquad \text{if} \quad x: s \in X$$

$$f_s^{\#}(\sigma(\mathbf{t})) = \sigma_A^{w,s'}(\mathbf{t}) \qquad \text{for some} \quad \sigma: w \to s' \leqslant s \text{ s.t. } \mathbf{t}: w \in T_{\mathrm{SIG}}(X).$$

We check that $f^{\#}$ is well defined:

(a) $t = x: s \in X$. $f_s^{\#}(x)$ is well defined as $f_s(x)$ is so, and we have $f_{s'}^{\#}(x) = f_{s'}(x) = f_{s''}(x) = f_{s''}^{\#}(x)$.

(b) $t = \sigma(\mathbf{t})$. Assume that $f_w^{\#}(\mathbf{t})$ is well defined for all $w \in S^*$ and that $f_w^{\#}(\mathbf{t}) = f_{w''}^{\#}(\mathbf{t})$ for all $w', w'' \in S^*$ s.t. $\mathbf{t}: w'$, $\mathbf{t}: w'' \in T_{\mathrm{SIG}}(X)$. If $\sigma(\mathbf{t}): s_1'$, $\sigma(\mathbf{t}): s_2' \in T_{\mathrm{SIG}}(X)$ there exist $\sigma: w_1 \to s_1 \leqslant s_1'$, $\sigma: w_2 \to s_2 \leqslant s_2'$ s.t. $\mathbf{t}: w_1$, $\mathbf{t}: w_2 \in T_{\mathrm{SIG}}(X)$. By assumption, $f_{w_1}^{\#}(\mathbf{t}) = f_{w_2}^{\#}(\mathbf{t}) \in A_{w_1} \cap A_{w_2}$. Then

$$f_{s_1'}^{\#}(\sigma^{w_1,s_1}(\mathbf{t})) = f_{s_1'}^{\#}(\sigma(\mathbf{t})) = \sigma_A^{w_1,s_1}(f_{w_1}^{\#}(\mathbf{t})) = \sigma_A^{w_2,s_2}(f_{w_2}^{\#}(\mathbf{t}))$$
$$= f_{s_2'}^{\#}(\sigma(\mathbf{t})) = f_{s_2'}^{\#}(\sigma^{w_2,s_2}(\mathbf{t})).$$

We conclude that $f^{\#}$ is well defined (use $s1 = s2$) and a SIG-homomorphism.

At last we prove unicity of the induced homomorphism. Let $h: T_{\mathrm{SIG}}(X) \to A \in \mathbf{OSALG}_{\mathrm{SIG}}$ s.t. $h_s(x) = f_s(x)$ for $x: s \in X$. Induction proves $f^{\#} = h$:

(a) $t = x: s \in X$. $h_s(x) = f_s(x) = f_s^{\#}(x)$.

(b) $t = \sigma(\mathbf{t})$. Assume that $f_w^{\#}(\mathbf{t}) = h_w(\mathbf{t})$ for all $w \in S^*$ s.t. $\mathbf{t}: w \in T_{\mathrm{SIG}}(X)$. Then

$$f_{s'}^{\#}(\sigma^{w,s}(\mathbf{t})) = f_{s'}^{\#}(\sigma(\mathbf{t})) = \sigma_A^{w,s}(f_w^{\#}(\mathbf{t})) = \sigma_A^{w,s}(h_w(\mathbf{t})) = h_{s'}(\sigma(\mathbf{t})) = h_{s'}(\sigma^{w,s}(\mathbf{t}))$$

for some $\sigma: w \to s \leqslant s'$ s.t. $\mathbf{t}: w \in T_{\mathrm{SIG}}(X)$. ∎

DEFINITION. An (order-sorted) *presentation* PRES consists of a signature SIG and a set E of equations of the form

$$[\mathbf{x}: w]\, t =_s t'$$

with $t, t': s \in T_{\mathrm{SIG}}(\{\mathbf{x}: w\})$ for some $s \in S$.

An equation $[\mathbf{x}: w]\, t =_s t'$ is *satisfied* by a SIG-algebra A if $h_s(\mathbf{t}) = h_s(\mathbf{t}')$ for all homomorphisms $h: T_{\mathrm{SIG}}(\{\mathbf{x}: w\}) \to A$.

A PRES-*algebra* is a SIG-algebra which satisfies all the equations of PRES. With SIG-homomorphisms this defines a category $\mathbf{OSAlg}_{\mathrm{PRES}}$.

Remarks. • As we deal with many-sorted algebras we must be careful about typing the equations [18]. We use schemes of the form

pres "name" **is**
sorts "sorts & ordering"
var "typing of variables"
eqns "equations"

and assume that each equation is only typed by the variables occurring. By this convention we forbid equations of the form $[x: s, \mathbf{x}: w]\, t =_s t'$, where x does not occur in t or t'.

• The same variable x may occur several times in $\mathbf{x}: w$ having different typings. Substitution replaces all occurrences of x by the same element which has to satisfy all type restrictions. For intuition it seems to be useful to allow the notation $x: s_1 \cap \cdots \cap s_n$.

• The conditions ensure that equations are restricted by sorting.

• We use "presentations" instead of "specifications" which will be subclass of presentations (cf. Section 3).

In an abstract, categorical framework PRES-algebras are "algebraic" over order-sorted sets, i.e., a left adjoint exists, limits are created as well as coequalizers of kernel pairs. One easily checks that categories of order-sorted sets have limits, colimits, and image factorization, hence $\mathbf{OSAlg}_{\mathrm{PRES}}$ has all the infrastructure to be hoped for in a category of algebras (for the terminology compare [20, 21]. We know this anyway because of the equivalence results above but the explicit construction turns out to be quite straightforward. In the present context we are only

interested in free constructions hence we leave a discussion of other infrastructure as an exercise.

As we do consider modularization techniques we have to deal with several presentations and presentation morphisms.

DEFINITION. A *morphism of signatures* $h: \mathrm{SIG} \to \mathrm{SIG}'$ consists of a monotone sort mapping $h: S \to S'$, i.e., $h(s) \leqslant h(s')$ if $s \leqslant s'$, and an $S^* \times S$-indexed function $h: \Sigma \to \Sigma'$ s.t.,

(i) $h_{w,s}: \Sigma_{w,s} \to \Sigma_{h(w),h(s)}$

(ii) $h_{w,s}(\sigma) = h_{w',s'}(\sigma)$ if $\sigma: w \to s$, $\sigma: w' \to s' \in \Sigma$.

2.5. LEMMA. *Every signature morphism* $h: \mathrm{SIG} \to \mathrm{SIG}'$ *induces a forgetful functor*

$$-_h: \mathbf{OSAlg}_{\mathrm{SIG}'} \to \mathbf{OSAlg}_{\mathrm{SIG}}, \qquad A \to A_h \text{ with } A_{h,s} := A_{h(s)} \text{ and } \sigma_{A_h}^{w,s} := \sigma_A^{h(w),h(s)}.$$

Proof. $A_{h,s} \subseteq A_{h,s'}$ for $s \leqslant s' \in S$ because $h(s) \leqslant h(s')$. For $\sigma: w \to s$, $\sigma: w' \to s' \in S$, $\mathbf{a}: w$, $\mathbf{a}: w' \in A_h$ we compute

$$\sigma_{A_h}^{w,s}(\mathbf{a}) = h(\sigma)_A^{h(w),h(s)}(\mathbf{a}) = h(\sigma)_A^{h(w'),h(s')}(\mathbf{a}) = \sigma_h^{w',s'}(\mathbf{a})$$

because $h(\sigma): h(w) \to h(s)$, $h(s): h(w') \to h(s') \in \Sigma$ and $\mathbf{a}: h(w)$, $\mathbf{a}: h(w') \in A$. ∎

For the definition of presentation morphisms renaming must be extended to terms.

2.6. LEMMA. *Let* $h: S \to S'$ *be a monotone mapping. Then* $Y_h := \{y: s \mid y: h(s) \in Y\}$ *defines a forgetful function* $-_h: \mathbf{Set}^{S'} \to \mathbf{Set}^S$.

For an order-sorted set X *we define* $h(X)_{s'} := \bigcup \{X_s \mid s \in S,\ h(s) \leqslant s'\}$. *With the embedding* $Xv: X \to h(X)_h$ *we obtain a free* S'*-set* $(h(X), Xv)$ *over* X.

The proof is straightforward.

DEFINITION. A morphism of presentations $h: \mathrm{PRES} \to \mathrm{PRES}'$ is a signature morphism $h: \mathrm{SIG} \to \mathrm{SIG}'$ of the underlying signatures such that

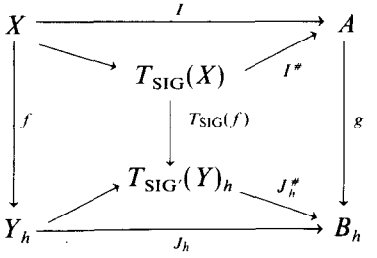
$$[\mathbf{x}: h(w)]\, h(t) =_{h(s)} h(t') \in E' \qquad \text{if} \quad [\mathbf{x}: w]\, t =_s t' \in E,$$

where the renaming $h: T_{\mathrm{SIG}}(\{\mathbf{x}: w\}) \to (T_{\mathrm{SIG}'}(\{\mathbf{x}: h(w)\}))_h \in \mathbf{OSAlg}_{\mathrm{SIG}}$ is defined by universal properties in the diagram

$$\begin{array}{ccc} \{\mathbf{x}: w\} & \xrightarrow{\quad\eta\quad} & T_{\mathrm{SIG}}(\{\mathbf{x}: w\}) \\ {\scriptstyle v}\downarrow & & \downarrow{\scriptstyle h} \\ h(\{\mathbf{x}: w\})_h = \{\mathbf{x}: h(w)\}_h & \xrightarrow[\eta_h]{} & T_{\mathrm{SIG}}(\{\mathbf{x}: h(w)\})_h \end{array}$$

The category of presentations and presentation morphisms is denoted by PRES. Subsequently, we assume a presentation morphism $h: \mathrm{PRES} \to \mathrm{PRES}'$ to be fixed.

2.7. LEMMA. *The diagram*

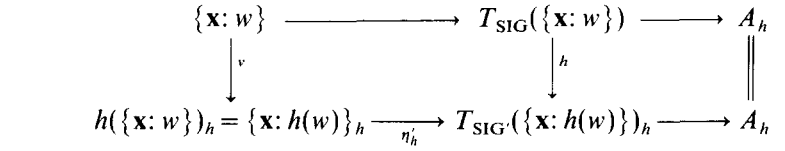


commutes if $g \circ I = J_h \circ f$.

Proof. By freeness of $T_{\mathrm{SIG}}(X)$. ∎

2.8. LEMMA. $-_h: \mathbf{OSAlg}_{\mathrm{PRES}'} \to \mathbf{OSAlg}_{\mathrm{PRES}}$, $A \to A_h$ *defines a forgetful functor.*

Proof. We only have to check that $[\mathbf{x}: w]\, t =_s t' \in E$ holds in A_h. Let $I: \{\mathbf{x}: w\} \to A_h$. Then $J: \{\mathbf{x}: h(w)\} \to A \in \mathbf{Set}^S$, where $J_{h(w)}(\mathbf{x}) = I_w(\mathbf{x})$. As A is a PRES'-algebra $[\mathbf{x}: h(w)]\, h(t) =_{h(s)} h(t')$ holds in A_h, and $I_s^\#(t) = J^\#_{h(s)}(h(t)) = J^\#_{h(s)}(h(t')) = I_s^\#(t')$ by Proposition 2.4 as $J_h \circ v = I$ in



∎

Free constructions are obtained by the standard strategy to factorize the term algebra by a suitable congruence relation. The notion of congruence reflects order-sorting.

DEFINITION [13]. An S-indexed relation $R = (R_s \subseteq A_s \times A_s \mid s \in S)$ is called a **SIG-congruence** if

- $R_s = R_s^* := \{(a, a') \in R^* \mid a: s,\ a': s \in A\}$ and
- $\sigma(\mathbf{a}) R^* \sigma(\mathbf{a}')$ for all $\sigma: w \to s$, $\sigma: w' \to s'$, $\mathbf{a}: w$, $\mathbf{a}': w' \in A$ such that $\mathbf{a} R^* \mathbf{a}'$,

where $R^* \subseteq A \times A$ is the least equivalence relation s.t. $\bigcup_{s \in S} R_s \subseteq R^*$ (here A denotes the unions of the respective indexed sets).

The *quotient* $A_{/R}$ is defined by

$$(A_{/R})_s := \{[a] \mid a: s \in A\}, \qquad \text{where} \quad [a] := \{a' \in A \mid a R^* a'\}$$

with operations being defined on the congruences as usual.

2.9. PROPOSITION. (i) $A_{/R}$ *is a SIG-algebra, and* $\pi: A \to A_{/R}$, $a \to [a]$ *is a homomorphism.*

(ii) *Given a homomorphism* $f: A \to B$ *such that*

$$R^* \subseteq \mathrm{Ker}(f) := \{(a, a') \in A \times A \mid f(a) = f(a')\}$$

then there exists a unique homomorphism $f': A_{/R} \to B$ *such that* $f = f' \circ \pi$.

(This observation is implicitly stated in [13].)

Proof. (i) Let $[a]: s \in A_{/R}$, $a: s \in A$, and $s \leqslant s'$. Then $a: s' \in A$ and $[a]: s' \in A_{/R}$. Moreover,

$$\sigma^{w,s}_{A\backslash R}([\mathbf{a}]) = [\sigma^{w,s}_A(\mathbf{a})] =^{(1)} [\sigma^{w',s'}_A(\mathbf{a}')] = \sigma^{w',s'}_{A\backslash R}([\mathbf{a}'])$$

for $\sigma: w \to s$, $\sigma: w' \to s' \in \Sigma$, $[\mathbf{a}]: w$, $[\mathbf{a}']: w' \in A_{/R}$. (1) holds by definition of congruences. $\pi_s(a) = [a] = \pi_{s'}(a)$ for $a: s$, $a: s' \in A$.

(ii) is a straightforward argument using $f'_s([a]) = f_s(a)$. ∎

In the present context we are only interested in free constructions for presentation embeddings PRES ⊆ PRES'.

DEFINITION. Given PRES-algebra A let an S'-set $\hat{A}$ be defined ambiguously by $\hat{A}_{s'} = \bigcup_{s \leqslant s'} A_s$. Let $t \approx t'$ be the smallest equivalence relation on $T_{\mathrm{SIG}'}(\hat{A})$ (here the union of the components of the S'-indexed set) s.t. $t \approx t'$ if

(a) $t = \sigma(\mathbf{a})$ and $t' = \sigma^{w,s}_A(\mathbf{a})$ for some $\sigma: w \to s \in \Sigma$ s.t. $\mathbf{a}: w \in \hat{A}$, or

(b) $t = I^{\#}_s(l)$, $t' = I^{\#}_s(r)$, where $[\mathbf{x}: w] l =_s r \in E'$ and $I: \{\mathbf{x}: w\} \to T_{\mathrm{SIG}'}(\hat{A})$, or

(c) $t = \sigma(\mathbf{t})$, $t' = \sigma(\mathbf{t}')$, where $\sigma: w \to s$, $\sigma: w' \to s' \in \Sigma$, $\mathbf{t}: w$, $\mathbf{t}': w' \in T_{\mathrm{SIG}}(\hat{A})$ s.t. $\mathbf{t} \approx \mathbf{t}'$.

Remark. Proofs involving $t \approx t'$ will invariably use a case analysis. Each time (a), (b), and (c) will refer to the respective cases of the definition.

2.10. LEMMA. *Let* $t \approx_s t'$ *if* $t \approx t'$ *and* $t: s$, $t': s \in T_{\mathrm{SIG}'}(\hat{A})$. *This defines a* SIG'*-congruence.*

Proof. Let $t \equiv t'$ be the least congruence containing $\bigcup_{s \in S} \approx_s$. Clearly $\equiv \subseteq \approx$. On the other hand, we prove $\approx \subseteq \equiv$ by inspection of cases:

(a) $\sigma(\mathbf{a}): s$, $\sigma^{w,s}_A(\mathbf{a}): s \in T_{\mathrm{SIG}'}(\hat{A})$, hence $\sigma(\mathbf{a}) \approx_S \sigma^{w,s}_A(\mathbf{a})$.

(b) Similarly $I^{\#}_s(l) \approx_s I^{\#}_s(r)$ as $l, r: s \in T_{\mathrm{SIG}'}(\hat{A})$.

(c) We assume that $\mathbf{t} \equiv \mathbf{t}'$ if $\mathbf{t} \approx \mathbf{t}'$. Then $\sigma(\mathbf{t}) \equiv \sigma(\mathbf{t}')$ by definition of congruences. ∎

2.11. PROPOSITION. $T_{\mathrm{PRES}'}(A) := T_{\mathrm{SIG}'}(\hat{A})_{/\approx}$ *is a free PRES'-algebra over the PRES-algebra* A. *The unit is* $A\eta: A \to (T_{\mathrm{PRES}'}(A))_{\mathrm{PRES}}$, $a \to [a]$.

(B_{PRES} denotes the PRES-component obtained from a PRES'-algebra under the forgetful functor, $\hat{A}$ is the underlying order-sorted set.)

Proof. It is easy to check that the unit is a homomorphism. Let $f: A \to B_{\mathrm{PRES}}$ be a PRES-homomorphism. Then there exists a unique extension $f^{\#}: T_{\mathrm{SIG}'}(\hat{A}) \to B$ in $\mathbf{OSAlg}_{\mathrm{SIG}'}$ (by composition of free constructions; observe that the S'-set $\hat{A}$ is free).

We check that $\approx \subseteq \mathrm{Ker}(f^{\#})$ by inspection of cases:

(a) $f^{\#}_s(\sigma(\mathbf{a})) = \sigma^{w,s}_B(f^{\#}_w(\mathbf{a})) = \sigma^{w,s}_B(f_w(\mathbf{a})) = f_s(\sigma^{w,s}_A(\mathbf{a})) = f^{\#}_s(\sigma^{w,s}_A(\mathbf{a}))$

(b) $f^{\#}_s(I^{\#}_s(l)) = f^{\#}_s(I^{\#}_s(r))$ using that $f^{\#} \circ I^{\#}: \{\mathbf{x}: w\} \to b$ and that B is a PRES′-algebra.

(c) $f^{\#}_s(\sigma(\mathbf{t})) = \sigma^{w,s}_B(f^{\#}_w(\mathbf{t})) = \sigma^{w',s'}_B(f^{\#}_{w'}(\mathbf{t})) = f^{\#}_{s'}(\sigma(\mathbf{t}))$, where $f^{\#}_w(\mathbf{t}) = f^{\#}_{w'}(\mathbf{t})$ is the inductive assumption.

Then 2.9(ii) yields the result. ∎

Remarks. • As order-sorted sets are algebras of a presentation with sorts only the construction yields the free functor $F: \mathbf{Set}^S \to \mathbf{OSAlg}_{\mathrm{PRES}}$, where S are the sorts of PRES.

• If we use the adjunction 2.6 in a canonical way we obtain a free construction for arbitrary presentation morphisms.

The differences to standard many-sorted algebras become obvious when we inspect the adjunction determined by a signature inclusion $\mathrm{SIG} = (S, \Sigma) \subseteq \mathrm{SIG}' = (S + S', \Sigma + \Sigma')$.

DEFINITION. Given a SIG-algebra A we construct a SIG′-algebra $T_{\mathrm{SIG}'}(A)$ by: For $t \in T_{\mathrm{SIG}'}(\hat{A})$ let $\mathbf{t}_A$ be inductively defined by

$$a_A := a \quad \text{if} \quad a \in \hat{A}$$

$$\sigma(\mathbf{t})_A := \begin{cases} \sigma^{w,s}_A(\mathbf{t}_A) & \text{if} \quad \mathbf{t}_A: w \in \hat{A} \text{ for some } \sigma: w \to s \in \Sigma \\ \sigma(\mathbf{t}_A) & \text{else.} \end{cases}$$

Moreover, let

$$T_{\mathrm{SIG}'}(A)_s := \{t_A \mid t: s \in T_{\mathrm{SIG}'}(\hat{A})\} \qquad \text{for} \quad s \in S$$

$$\sigma^{w,s}_{T_{\mathrm{SIG}'}(A)}: T_{\mathrm{SIG}'}(A)_w \to T_{\mathrm{SIG}'}(A)_s,\ \mathbf{t}_A \to \sigma(\mathbf{t})_A \qquad \text{for} \quad \sigma: w \to s \in \Sigma + \Sigma',\ \mathbf{t}': w \in T_{\mathrm{SIG}'}(\hat{A}).$$

($T_{\mathrm{SIG}'}(\hat{A})$ is the term algebra generated by the order-sorted set $\hat{A}_{s'} = \bigcup_{s \leqslant s'} A_s$, $s' \in S$.)

2.12. PROPOSITION. (i) $-_A: T_{\mathrm{SIG}'}(\hat{A}) \to T_{\mathrm{SIG}'}(A)$ *is surjective.*

(ii) $T_{\mathrm{SIG}'}(A)$ *is a free SIG′-algebra over* A.

Proof. The operations are well defined and satisfy the requirements of SIG′-algebras as the definitions only depend on names.

(i) Surjectivity follows from the following observation. Let $\sigma: w \to s \in \Sigma + \Sigma'$, $\mathbf{t}: w \in T_{\mathrm{SIG}'}(\hat{A})$. We distinguish two cases:

(a) $\mathbf{t}_A: w' \in \hat{A}$ for some $\sigma: w' \to s' \in \Sigma$:

$$\sigma^{w,s}_{T_{\mathrm{SIG}'}(\hat{A})}(\mathbf{t})_{A,s} = \sigma(\mathbf{t})_A = \sigma^{w',s'}_A(\mathbf{t}_A) = \sigma^{w,s}_{T_{\mathrm{SIG}'}(\hat{A})}(\mathbf{t}_{A,w}).$$

(Observe that $t_{A,s} = t_A: s \in T_{\mathrm{SIG}'}(A)$ by definition.)

(b) Otherwise the argument is straightforward.

(ii) Embedding of A is the unit. Let $f: A \to B_{\mathrm{SIG}}$ be a SIG-homomorphism. We define the extension $f^{\#}: T_{\mathrm{SIG}'}(A) \to B$ by

$$f^{\#}_s(a) := f_s(a) \qquad \text{if} \quad a: s \in \hat{A}$$

$$f^{\#}_s(\sigma(\mathbf{t})) := \sigma^{w',s'}_B(f^{\#}_w(\mathbf{t})) \qquad \text{if} \quad \sigma: w' \to s' \in \Sigma + \Sigma',\ \mathbf{t}: w' \in T_{\mathrm{SIG}'}(A).$$

"Homomorphism conditions":

(a) $\mathbf{t} = \mathbf{a}: w \in A$ for some $\sigma: w \to s \in \Sigma$:

$$\begin{aligned} f^{\#}_s(\sigma^{w,s}_{T_{\mathrm{SIG}'}(A)}(\mathbf{a})) &= f^{\#}_s(\sigma^{w',s'}_A(\mathbf{a})) \\ &= f_{s'}(\sigma^{w',s'}_A(\mathbf{a})) \qquad \text{(by definition of } f^{\#}\text{)} \\ &= \sigma^{w',s'}_B(f_{w'}(\mathbf{a})) \\ &= \sigma^{w,s}_B(f^{\#}_w(\mathbf{a})) \qquad \text{(as } B \text{ is a SIG}'\text{-algebra and by definition of } f^{\#}\text{)} \end{aligned}$$

(b) Otherwise $f^{\#}_s(\sigma^{w,s}_{T_{\mathrm{SIG}'}(A)}(\mathbf{t})) = f^{\#}_s(\sigma(\mathbf{t})) = \sigma^{w,s}_B(f^{\#}_w(\mathbf{t}))$.

"Uniqueness" follows from the by-now-standard case distinction. ∎

Order-sorting causes existence of (sub-)terms of the form $t = \sigma(\cdots\sigma'(\cdots)\cdots)$: $s \in T_{\mathrm{SIG}'}(A)$, where $\sigma \in \Sigma$ and $\sigma' \in \Sigma'$ and $s \notin S$. As a consequence such a term t is not necessarily equivalent to an element of A in $T_{\mathrm{SPEC}'}(A)$ if the unit $\eta: A \to (T_{\mathrm{SPEC}'}(A))_{\mathrm{SPEC}}$ is an isomorphism. This is a major deviation to standard many-sorted algebra which will deserve some attention if parametrization is discussed.

The construction of free algebras does not work for the order sorted algebras of Goguen and Meseguer [19] (compare Section 2.1): Given the regular (!) signatures

$$(s, s', f: s \to s, f: s' \to s') \subseteq (s \leqslant s'',\ s' \leqslant s'',\ f: s \to s,\ f: s' \to s')$$

our construction may cause identifications, e.g., of b and c if $f(a) = b$ of type s and $f(a) = c$ of type s' assuming that a and b (resp. a and c) are the only data of type s (resp. s'). But the free algebra must distinguish b and c.

There are two alternatives: either to disambiguate the target signature as suggested in 2.3, and then to apply the construction above, or to disambiguate even further in that all operators come with type information and then to factorize by

a congruence relation where essentially condition (c) of the congruence $t \approx t'$ is replaced by

(c') $t = \sigma^{w,s}(\mathbf{t})$, $t' = \sigma^{w',s'}(\mathbf{t}')$, where $\sigma: w \to s \leqslant \sigma: w' \to s' \in \Sigma$, $\mathbf{t}: w$, $\mathbf{t}': w' \in A$ s.t. $\mathbf{t} \approx \mathbf{t}'$.

Both alternatives somewhat lack the lucidity of representation which is the goal of the whole undertaking. In case all generators have a least sort our construction works. Thus one may not be too worried about the algebras one is interested in as for those all the elements have a least sort. Such algebras are obtained by constructions such as the free algebra construction, forgetting along a presentation morphism, and maybe pushout of algebras (provided that least sorts are "preserved") and full abstraction, at least as long as only regular signatures are involved. These constructions never generate a situation where the "same" operator applied to the "same" data yields a different result. This is obvious for forgetful functors and follows immediately for the construction of free algebras as regularity implies that all elements of the term algebra have a least sort if the generators do [19]. As a consequence for these kind of algebras which are "term generated" in a wider sense, there is no difference whatsoever between the approaches of [19] and the one pursued here.

However, I believe there are arguments in favour of the approach chosen here:

- I reiterate that ad hoc polymorphism does not enhance the lucidity of specifications.
- The remarks above show that no sacrifices are made except to increase the specification discipline.
- The simple paradigm that the same operator applied to the same data yields the same result.
- No technical restrictions such as regularity which tend to blow up specifications.

One may object to the last point that regularity induces a minimal type for each term which is convenient for purposes of parsing (cf. [17]). But minimal types are given implicitly in our approach via "conjunction sorts" (compare Section 2.1) the handling of which may be safely left with the compiler/interpreter.

2.3. *Order-Sorted Theories*

The proof system has to be taylored to deal with overloading. Let SIG be a signature and X be a set of variable(name)s. *Formulas* are of the form $[\mathbf{x}: w]\, t =_s t'$ with $t, t': s \in T_{\mathrm{SIG}}(\{\mathbf{x}: w\})$ or $[\mathbf{x}: w]\, t = t'$ with $t, t' \in T_{\mathrm{SIG}}(\{\mathbf{x}: w\})$.

RULES.

Sort widening. $[\mathbf{x}: w]\, t =_s t' \vdash [\mathbf{x}: w]\, t = t'$

Sort narrowing. $[\mathbf{x}: w]\, t = t' \vdash [\mathbf{x}: w]\, t =_s t'$ if $t, t': s \in T_{\mathrm{SIG}}(\{\mathbf{x}: w\})$

Compatibility. $[\mathbf{x}: w]\, \mathbf{t} = \mathbf{t}' \vdash [\mathbf{x}: w]\, \sigma(\mathbf{t}) = \sigma(\mathbf{t}')$ if $\sigma: w \to s$, $\sigma: w' \to s' \in \Sigma$, $\mathbf{t}: w$, $\mathbf{t}': w' \in T_{\mathrm{SIG}}(\{\mathbf{x}: w\})$

Substitutivity. $[\mathbf{x}: w]\, t = t' \vdash [\mathbf{x}': w']\, t[\mathbf{x}/\mathbf{t}''] = t[\mathbf{x}/\mathbf{t}'']$ if $\mathbf{t}'': w \in T_{\mathrm{SIG}}(\{\mathbf{x}': w'\})$

Shuffling. $[\mathbf{x}: w, x: s, y: s', \mathbf{v}: w']\, t = t' \vdash [\mathbf{x}: w, y: s', x: s, \mathbf{v}: w']\, t = t'$

Equivalence. $\vdash [\mathbf{x}: w]\, t = t$,

$$[\mathbf{x}: w]\, t = t' \vdash [\mathbf{x}: w]\, t' = t$$

$$[\mathbf{x}: w]\, t = t',\ [\mathbf{x}: w]\, t' = t'' \vdash [\mathbf{x}: w]\, t = t'',$$

where $t[\mathbf{x}/\mathbf{t}']$ states that $\mathbf{t}'$ is simultaneously substituted for $\mathbf{x}$ in t.

Remark. Overloading incurs sort widening and narrowing and shuffling, the latter not a subcase of substitution, as the same variable may occur several times in $\mathbf{x}: w$.

DEFINITION. Given a presentation PRES we say that $[\mathbf{x}: w]\, t =_{(s)} t'$ is *derivable* from PRES if $[\mathbf{x}: w]\, t =_{(s)} t'$ can be deduced from the PRES-equations using the deduction rules above. Notation is $\vdash_{\mathrm{PRES}} [\mathbf{x}: w]\, t =_{(s)} t'$.

A SIG-algebra *satisfies* $[\mathbf{x}: w]\, t' =_{(s)} t''$ if $I^{\#}_{s'}(t') = I^{\#}_{s''}(t'') \in A_{s'} \cap A_{s''}$ for all interpretations $I: \{\mathbf{x}: w\} \to A \in \mathbf{Set}^S$ and all $s', s'' \in S$ s.t. $t': s'$, $t'': s'' \in T_{\mathrm{SIG}}(\{\mathbf{x}: w\})$ (this definition extends the one given in 2.2).

$[\mathbf{x}: w]\, t =_{(s)} t'$ is a *consequence* of PRES if all PRES-algebras satisfy $[\mathbf{x}: w]\, t =_{(s)} t'$. We use the standard notation $\models_{\mathrm{PRES}} [\mathbf{x}: w]\, t =_{(s)} t'$.

2.13. PROPOSITION. *Deduction is sound; i.e.,* $\vdash_{\mathrm{PRES}} [\mathbf{x}: w]\, t =_{(s)} t'$ *implies* $\models_{\mathrm{PRES}} [\mathbf{x}: w]\, t =_{(s)} t'$.

Proof. Let A be a SIG-algebra. We use distinction by cases (the data of the rules are used correspondingly):

Widening. If A satisfies $[\mathbf{x}: w]\, t' =_s t''$ then $I^{\#}_{s'}(t') = I^{\#}_{s}(t') = I^{\#}_{s}(t'') = I^{\#}_{s''}(t'')$ by definition of homomorphisms where $t': s'$, $t'': s'' \in T_{\mathrm{SIG}}(\{\mathbf{x}: w\})$.

Narrowing. Follows from the definition of satisfaction.

Shuffling. The order of the variables is irrelevant w.r.t. the definition of satisfaction.

Compatibility. If A satisfies $[\mathbf{x}: w]\, \mathbf{t}' = \mathbf{t}''$ we have $I^{\#}_{w'}(\mathbf{t}') = I^{\#}_{w''}(\mathbf{t}'') \in A_{w'} \cap A_{w''}$. Then $I^{\#}_{s'}(\sigma(\mathbf{t}')) = \sigma_A^{w',s'}(I^{\#}_{w'}(\mathbf{t}')) = \sigma_A^{w'',s''}(I^{\#}_{w''}(\mathbf{t}'')) = I^{\#}_{s''}(\sigma(\mathbf{t}''))$ using the properties of homomorphisms and algebras.

Substitutivity. We define a mapping sub: $T_{\mathrm{SIG}}(\{\mathbf{x}: w\}) \to T_{\mathrm{SIG}}(\{\mathbf{x}': w'\})$ by $\mathrm{sub}_w(\mathbf{x}) = \mathbf{t}$ (this is well defined as the order-sorted set $\{\mathbf{x}: w\}$ is generated by $\mathbf{x}: w$). We check that $I^{\#}(t[\mathbf{x}/\mathbf{t}]) = I^{\#} \circ \mathrm{sub}(t)$ for every $t \in T_{\mathrm{SIG}}(\{\mathbf{x}: w\})$:

(a) $t = x_i: s_i \in \{\mathbf{x}: w\}$: Then $I^{\#}_{s_i}(x_i[\mathbf{x}/\mathbf{t}]) = I^{\#}_{s_i}(t_i) = I^{\#}_{s_i}(\mathrm{sub}_{s_i}(x_i))$.

(b) $t = \sigma(\mathbf{t}_1)$ with $\sigma: w_1 \to s_1 \leqslant s_2 \in \Sigma$ and $\mathbf{t}_1 : w_1 \in T_{\mathrm{SIG}}(\{\mathbf{x}': w'\})$: Inductive assumption is $I^{\#}_{w_1}(\mathbf{t}_1[\mathbf{x}/\mathbf{t}]) = I^{\#}_{w_1} \circ \mathrm{sub}_{w_1}(\mathbf{t})$. Then

$$\begin{aligned} I^{\#}_{s_2}(\sigma(\mathbf{t}_1)[\mathbf{x}/\mathbf{t}]) &= I^{\#}_{s_2}(\sigma(\mathbf{t}_1[\mathbf{x}/\mathbf{t}])) = I^{\#}_{s_2}(\sigma^{w_1,s_1}(\mathbf{t}_1[\mathbf{x}/\mathbf{t}])) \\ &= \sigma_A^{w_1,s_1}(I^{\#}_{w_1}(\mathbf{t}_1[\mathbf{x}/\mathbf{t}])) = \sigma_A^{w_1,s_1}(I^{\#}_{w_1} \circ \mathrm{sub}_{w_1}(\mathbf{t}_1)) \\ &= I^{\#}_{s_1} \circ \mathrm{sub}_{s_1}(\sigma^{w_1,s_1}(\mathbf{t}_1)) = I^{\#}_{s_2} \circ \mathrm{sub}_{s_2}(\sigma(\mathbf{t}_1)). \end{aligned}$$

Thus we can compute

$$I^{\#}_{s'}(t'[\mathbf{x}/\mathbf{t}]) = I^{\#}_{s'} \circ \mathrm{sub}_{s'}(t') = I^{\#}_{s''} \circ \mathrm{sub}_{s''}(t'') = I^{\#}_{s''}(t''[\mathbf{x}/\mathbf{t}]),$$

where we use that $[\mathbf{x}: w]t' = t''$ holds in A.

Equivalence. Straightforward. ▮

2.14. PROPOSITION. *Deduction is complete*; *i.e.*, $\models_{\mathrm{PRES}} [\mathbf{x}: w]t =_{(s)} t'$ implies $\vdash_{\mathrm{PRES}} [\mathbf{x}: w]t =_{(s)} t'$.

Proof. As in general there is no algebra being "functionally free" (compare [18]) we use algebras which are "functionally free w.r.t. a certain type of equations" (i.e., algebras in which equations of a certain type only hold if they are derivable) to check completeness.

We prove that $\vdash_{\mathrm{PRES}} [\mathbf{x}: w]t =_{(s)} t'$ if $[\mathbf{x}: w]t =_{(s)} t'$ is satisfied by $T_{\mathrm{PRES}}(\{\mathbf{x}: w\})$: Using the canonical factorization $\pi: T_{\mathrm{SIG}}(\{\mathbf{x}: w\}) \to T_{\mathrm{PRES}}(\{\mathbf{x}: w\}$ we conclude that $t \approx t'$. Thus we have to check that $\vdash_{\mathrm{PRES}} [\mathbf{x}: w]t =_{(s)} t'$ if $t \approx t'$:

(a) Since $t = t' = \sigma(\mathbf{a})$.

(b) $t = I^{\#}_{s'}(l)$, $t' = I^{\#}_{s'}(r)$ with $[\mathbf{x}': w']l =_{s'} r \in E$ and $I: \{\mathbf{x}': w'\} \to T_{\mathrm{SIG}}(\{\mathbf{x}: w\})$. We compute

$$\begin{aligned} &\vdash_{\mathrm{PRES}} [\mathbf{x}': w']l =_{s'} r \\ &\vdash_{\mathrm{PRES}} [\mathbf{x}': w']l = r && \text{(widening)} \\ &\vdash_{\mathrm{PRES}} [\mathbf{x}: w]\, l[\mathbf{x}'/I^{\#}_{w'}(\mathbf{x}')] = r[\mathbf{x}'/I^{\#}_{w'}(\mathbf{x}')] && \text{(substitutivity)} \\ &\vdash_{\mathrm{PRES}} [\mathbf{x}: w]\, l[\mathbf{x}'/I^{\#}_{w'}(\mathbf{x}')] =_{s'} r[\mathbf{x}'/I^{\#}_{w'}(\mathbf{x}')] && \text{(narrowing)} \\ &\vdash_{\mathrm{PRES}} [\mathbf{x}: w] I^{\#}_{w'}(l) =_{s'} I^{\#}_{w'}(r); \end{aligned}$$

for the last step we use that $I^{\#}$ is a SIG-homomorphism.

(c) $t' = \sigma(\mathbf{t}')$, $t'' = \sigma(\mathbf{t}'')$ with $\sigma: w' \to s'$, $\sigma: w'' \to s'' \in \Sigma$ s.t. $\mathbf{t}': w'$, $\mathbf{t}'': w'' \in T_{\mathrm{SIG}}(\{\mathbf{x}: w\})$ and $\mathbf{t}' \approx \mathbf{t}''$. The inductive assumption is $\vdash_{\mathrm{PRES}} [\mathbf{x}': w']\mathbf{t}' = \mathbf{t}''$. But then compatibility ensures $\vdash_{\mathrm{PRES}} [\mathbf{x}': w']\, \sigma(\mathbf{t}') = \sigma(\mathbf{t}'')$.

(d) The non-trivial case is transitivity. Let $[\mathbf{x}: w]t = t'$ and $[\mathbf{x}': w']t' = t''$ by inductive assumption, where $\mathbf{x}': w$ is a permutation of $\mathbf{x}: w$. Then $[\mathbf{x}: w]t' = t''$ by shuffling. ▮

Remark. A deduction system for the notion of order-sorted algebras as in [19] is obtained if we replace the compatibility rule by

$$[\mathbf{x}: w]\mathbf{t} = \mathbf{t}' \vdash [\mathbf{x}: w]\ \sigma(\mathbf{t}) = \sigma(\mathbf{t}')$$

if $\sigma: w \to s \leqslant \sigma: w' \to s' \in \Sigma$, $\mathbf{t}: w$, $\mathbf{t}': w' \in T_{\mathrm{SIG}}'(\{\mathbf{x}: w\})$.

2.4. *Order-Sorted Specifications as Data Specifications*

The results given so far indicate that specifications with subsorts are formally well based. We have, however, to ask whether they are a suitable tool for specification. We have used subsorts to avoid "errors" in a specification such as

pres STACK **is**
sorts $*data*, *nestack* < *stack*
ops empty: → *stack*
push: *stack data* → *nestack*
pop: *nestack* → *stack*
top: *nestack* → *data*
var *s*: *stack*, *d*: *data*
eqns pop(push(*s*, *d*)) = *s*
top(push(*s*, *d*)) = *d*

($ indicates "parameter data") the rationale being that error terms such as "pop(empty)" cannot be generated due to syntactical restrictions. This appears to be a most elegant way to handle errors by elimination. One does, however, not gain without losing; the syntactical restrictions do not allow formation of terms such as "pop(pop(push(push(empty, *d*), *d*')))" which are reasonable in that application of equations yields an "equivalent" term "pop(push(empty, *d*)" which is syntactically correct.

Hence a grain of partiality is introduced in that one would rather consider the standard language of stacks but allow that not every term has a denotation. One may split a module into two components, the *language specification* and the *data specification* which are linked by an *interface*, e.g.,

spec STACK **is**
syntax **sorts** $*DATA*, *STACK*
ops empty: → *STACK*
push: *STACK DATA* → *STACK*
pop: *STACK* → *STACK*
top: *STACK* → *DATA*
data **sorts** $*data* < *DATA*, *nestack* < *stack* < *STACK*
ops empty: → *stack*
push: *stack data* → *nestack*
pop: *nestack* → *stack*
top: *nestack* → *data*
var *s*: *stack*, *d*: *data*
eqns pop(push(*s*, *d*)) = *s*
top(push(*s*, *d*)) = *d*

Terms then may be called *admissable* if they are syntactically correct with regard to the *syntax* signature. Admissable terms, however, may have no denotation in that there is no "equivalent" term which is *well-typed*, i.e., syntactically correct with regard to the *data* signature. Roughly, terms are equivalent if they can be transformed into each other by correct application of the equations to subterms, e.g.,

$$\text{pop}(\text{pop}(\text{push}(\text{empty}, d))) = \text{pop}(\text{empty}) \qquad \text{is not defined,}$$

but

$$\text{pop}(\text{pop}(\text{push}(\text{push}(\text{empty}, d), d'))) = \text{pop}(\text{push}(\text{empty}, d)) = \text{empty} \qquad \text{is defined}$$

(proviso d, d': *data*). To refer to this mechanism as *run-time type checking* seems justified as, thinking of animation by rewriting, a (partial) computation of the result is the prerequisite to decide if a term denotes some data or prompts an error message. On the other hand, analysis of well-formed terms clearly is a compile time activity. This point of view is implicitly states as well in [17].

The intuition the example provides is restricted to initial structures. It is not so straightforward to find the appropriate notion of models of such specification. We tackle this problem in [26] where we consider subsorting in the framework of partial algebras. The point to be made here is that specifications with subsorts are a tool to specify data but do not necessarily provide a language to talk about the data.

3. Parametrization with Subsorts

3.1. *Parametrization Revisited*

The theory of *abstract data types* relies on at least two paradigms:

- separation of specification and implementation, of the "what" and "how," and
- modularization as a structuring discipline.

Modularization is motivated pragmatically in that big programming systems can be split into comprehensible units, thus easing the task of the designer and of the user. Modules should be thought of "pieces of software" being independent of any environment in which they occur. These paradigms have several consequences when the interaction of modules is considered. For instance, we may take "usage" as an elementary relation between modules stating that the facilities offered by module B can be used in a module A. Given *specifications* of module A and B "independence of the environment" may be interpreted to mean that A and B can be implemented separately and that the *implementation* of A can "use" the *implementation* of B without restrictions.

Let us be more precise: if we identify modules with standard many-sorted

specifications[5] "usage" naturally relates to the notion of subspecification: SPEC is a *subspecification* of SPEC′ if

$$\text{SPEC}' = \text{SPEC} + (S', \Sigma', E') := (S + S', \Sigma + \Sigma', E + E')$$

(SPEC′ is also called a *combination* of SPEC and (S', Σ', E')). We assume that the initial algebras provide the semantics of the respective specifications. The export of the specification SPEC consits of its signature. The operators of SPEC may be used, in combination with those of SPEC′, to generate terms which may happen to be of a sort $s \in S$ imported from SPEC.

EXAMPLE.

spec NAT **is**
sorts *nat*
ops 0: → *nat*
suc: *nat* → *nat*
add: *nat nat* → *nat*
var *m*, *n*: *nat*
eqns add(*m*, 0) = *m*
add(*m*, suc(*n*)) = suc(add(*m*, *n*))

spec NATSTACK **is**
NAT **with**
sorts *stack*
ops empty: → *stack*
push: *stack nat* → *stack*
pop: *stack* → *stack*
top: *stack* → *nat*
var *s*: *stack*, *m*: *nat*
eqns pop(push(*s*, *m*)) = *s*
top(push(*s*, *m*)) = *m*

NATSTACK uses NAT; however, the specifications are not independent; "top(empty)" is a term which is supposed to denote a "natural number" but fails to do so.

In order to allow for independence of implementations each term of sort $s \in S$ must have a denotation determined by SPEC alone; otherwise such terms cannot be interpreted in every (correct) implementation of SPEC. Moreover, one would expect from a logical point of view that no additional facts about SPEC-data can be proven in SPEC′; i.e., SPEC′ is a consistent or conservative extension of SPEC. As every SIG-term can be conceived as a SIG′-term our requirements for *correctness* of the usage relation can be stated by:

Sufficient completeness. $t =_{E+E'} t'$, for some $t' \in T_{\text{SIG}}$ if $t \in T_{\text{SIG}',s}$ with $s \in S$

Consistency. $t =_E t'$, if $t =_{E+E'} t'$ for $t, t' \in T_{\text{SIG}',s}$ with $s \in S$.

[5] We consider the simplest form of modules for sake of the argument. Consult [9] for a more sophisticated view.

Adding the equations "pop(empty) = empty" and "top(empty) = 0" to the specification NASTACK guarantees these requirements, though by brute force.

Parametrization has been suggested as an operator to build modules from given modules [3, 4, 6, 8]. Parametrization aims for "reusability" of software in that a module is defined relative to a formal parameter (specification) which can be replaced by an actual parameter (module). A *parametrized specification* is combination of a *formal parameter* PSPEC = $(PS, P\Sigma, PE)$ and a *body* $(BS, B\Sigma, BE)$. Typical examples are the specifications of stacks, arrays, queues, etc., where the parameter states the kind of data to be handled. The simplest form of *parameter passing* replaces the formal parameter by an *actual parameter* ASPEC = $(AS, A\Sigma, AE)$ of which PSPEC is a subspecification. Then RSPEC = ASPEC + $(BS, B\Sigma, BE)$ is the *result specification.*

EXAMPLE.

spec STACK **is**
sorts $\$data$, *stack*
ops empty: $\rightarrow$ *stack*
push: *stack data* $\rightarrow$ *stack*
pop: *stack* $\rightarrow$ *stack*
top: *stack* $\rightarrow$ *data*
var *s*: *stack*, *d*: *data*
eqns pop(push(*s*, *d*)) = *s*
top(push(*s*, *d*)) = *d*

Updating with NAT yields NATSPEC. We use "$" to indicate the formal parameter.

RSPEC obviously uses ASPEC but also uses the implementation of the parametrized specification. An implementation of a parametrized data type should be thought of as a construction which, given the parameter passing information, provides an implementation of the result specification based on the implementation of the actual parameter only. For instance, such a construction would give a stack implementation, say by lists, for every type of data, with the only information being passed being the type of data. The underlying assumption of independence of the actual parameter and the parametrized specification, however, is wrong: the combination may generate data, an implementation of which is not provided, e.g., the terms "top(empty)" and "push(empty, suc(top(empty)))." Either the result specification does not correctly use the actual parameter or it does not correctly use the body. The latter needs to be explained more precisely:

The implementation of the parametrized specification can rely on those components of an actual parameter which are matched to the data of the formal parameter. Presence of additional sorts and operators should cause no difference in what is constructed in order to ensure uniformity of the construction. The stack example, however, demonstrates that data are generated by the result specification, namely "push(empty, suc(top(empty)))," which are not obtained if the stack is just built over natural numbers. Reference [8] introduces the criterion of "passing

compatibility" in order to guarantee that implementations of a parametrized specification do not have to worry about unwanted interference by an actual parameter. Formally, parameter passing is modelled by pushouts in the category of presentations

$$\begin{array}{ccc} \text{PSPEC} & \underset{p}{\subseteq} & \text{SPEC} \\ {\scriptstyle h}\downarrow & & \downarrow{\scriptstyle h'} \\ \text{ASPEC} & \underset{p'}{\subseteq} & \text{RSPEC} \end{array}$$

Passing compatibility states that $F_p \circ V_h \approx V_{h'} \circ F_{p'}$ (F_p, $F_{p'}$ are free constructions and V_h, $V_{h'}$ are forgetful functors. We consider the case that ASPEC is not parametrized). Passing compatibility and *parameter protection*, i.e., correct usage of the actual parameter, then allow combination of the implementations of parametrized specification and the actual parameter without restriction. Both criteria are exactly satisfied if the parametrized specification is *persistent*; i.e., parameter protection holds for the formal parameter. This is the message of the "extension lemma" in [8]. In fact, passing compatibility implies persistency except for a degenerate case [22].

A less addressed consequence of modularization is that responsibility of design and implementation is distributed and that bugs and errors can be exactly attached to a specific module or subset of modules. This assumes that properties of a module which are established at some stage of the design process are preserved by operations and relations on modules simply in order to keep responsibility traceable. The correctness criteria ensure "preservation of properties." They may be seen as automated proof procedures which generate correct specifications under the assumption that correct specifications are given as input.

Though well justified, the correctness criteria appear sometimes to be too restrictive if not counterproductive, the foremost example being handling of errors.

EXAMPLE.

spec ARRAY
BOOL **with**
sorts $data, $index, *array*
ops $eq: *index index* → *bool*
new: → *array*
get: *array index* → *data*
update: *array index data* → *array*
if_then_else_: *bool array array* → *array*
if_then_else_: *bool data data* → *data*
$derror: → *data*
$okdata: *data* → *bool*
aerror: → *array*
okarray: *array* → *bool*
var *a*, *a'*: *array*, *i*, *j*, *k*: *index*, *d*, *d'*: *data*

eqns $eq(i, i) = true
$eq(i, j) = eq(j, i)
$eq(i, j) and eq(j, k) = eq(i, k)
$okdata(derror) = false
get(update(a, i, d), j) = if eq(i, j) and okarray(a) then d else get(a, j)
update(update(a, i, d), j, d') = if eq(i, j) and okarray(a)
then update(a, j, d')
else update(update(a, j, d'), i, d)
get(new, i) = get(aerror, i) = derror
update(aerror, i, d) = update(a, i, derror) = aerror
okarray(new) = true
okarray(update(a, i, d)) = okarray(a) and okdata(d)
okarray(aerror) = false
if true then a else a' = a
if false then a else a' = a'
if true then d else d' = d
if false *then* d else d' = d'

where BOOL is a shared specification of Booleans providing the canonical operators; "derror" has to be part of the parameter bidding for persistency. Moreover, parts of the error handling, namely, the definition of the ok-predicates [1], has to be provided by the actual parameter. This phenomenon is rather awkward as the error handling should be an inherent part of the specification of arrays which cannot be anticipated by the designer of a specification which happens to be an actual parameter (which is basically any specification regarding the widespread use of arrays). The error handling should be confined to the module it stems from in order to retain "true" modularization where a specification does not reflect all the contexts in which it may be used. The only alternative is to add a suitable error handling to an actual parameter but with the obvious disadvantage that consistency of the extension is to be proved in each case. Even then the proceeding does not exactly agree with the spirit of modularization as the error handling is definitely caused by the ARRAY-module, hence it should be the task of the designer of ARRAY to prove appropriate facts about the array and especially the error handling.

In fact, the situation is even worse as ARRAY is by no means persistent; if both the data and the index component are updated by the same sort, say *nat*, an error element must be added to the natural numbers which causes an error on indexes. Hence one has to add equations "update(*a*, inerror, *d*) = aerror" and "get(*a*, inerror) = derror" and to relativise all the other equations. But this intuition works properly only if updating is restricted to identifying the errors of the index and the data component as otherwise a plethora of "errors" is generated. These side effects caused by persistency are almost intractable and do certainly not enhance lucidity of specifications.

Subsorting at least allows for a less baroque notation

spec ARRAY
BOOL **with**
sorts $*data* < *data*+$, $*$index*, *array* < *array*+$

ops $eq: *index index* → *bool*
new: → *array*
get: *array*+ *index* → *data*
update: *array index data* → *array*
update: *array*+ *index data*+ → *array*+
if_then_else_: *bool array*+ *array*+ → *array*+
if_then_else_: *bool data*+ *data*+ → *data*+
derror: → *data*+
aerror: → *array*+

var *a*: *array*, *i*, *j*, *k*: *index*, *d*, *d'*: *data*, *a'*, *a''*: *array*+, *d''*, *d'''*: *data*+

eqns $eq(*i*, *i*) = true
$eq(*i*, *j*) = eq(*j*, *i*)
$eq(*i*, *j*) and eq(*j*, *k*) = eq(*i*, *k*)
get(update(*a*, *i*, *d*), *j*) = if eq(*i*, *j*) then *d* else get(*a*, *j*)
update(update(*a*, *i*, *d*), *j*, *d'*) = if eq(*i*, *j*) then update(*a*, *j*, *d'*)
else update(update(*a*, *j*, *d'*), *i*, *d*)
get(new, *i*) = get(aerror, *i*) = derror
update(aerror, *i*, *d''*) = update(*a'*, *i*, derror) = aerror
if true then *a'* else *a''* = *a'*
if false then *a'* else *a''* = *a''*
if true then *d''* else *d'''* = *d''*
if false then *d''* else *d'''* = *d'''*

Inspection of the specification proves that no data are generated on the parameter sorts which are not passed over from the actual parameter and that passing compatibility holds as well. Thus the parametrized specification can be considered independently of any environment it may be used in. Clearly, then there is the additional task to implement the "new sorts" *data*+ and *array*+ which is easily accomplished by adding an error component by a union construct.

These observations have been the motivation to consider subsorting in [23, 24]. Intuitively, we are able to abandon the condition of parameter protection using subsorts as we can add data to a parameter sort (though this is not true technically, see below). In fact, we do not exactly achieve our goal as terms such as "add(3, get(update(new, 1, 2), 1))" in ARRAY(NAT) as syntactically incorrect while semantically meaningful. The problem is the same as that pointed out in Section 2.4, namely, that specifications with subsorts may be thought of as data specifications leaving out the question of an appropriate language to address the data. The following discussion of parametrization for specification with subsorts should be seen under the restriction to consider "data specifications" only.

3.2. *Examining Parameter Passing*

We assume as a working hypothesis that a parametrized specification PSPEC ⊆ SPEC is an inclusion of presentations. We analyze several examples using (the obvious) pushouts in the category of presentations and demonstrate the need for a more restricted notion of parametrized specification.

A first observation is that morphisms of presentations are not well behaved

semantically in that the forgetful functors induced do not preserve "meaning" as demonstrated by a simple example

pres ONE **is**
sorts s, s'
ops $a: \to s$
$f: s' \to s'$

pres TWO **is**
sorts s, s'
ops $a: \to s$
$f: s \to s$
$f: s' \to s'$
eqns $f(a) = a$

The initial TWO-algebra has carriers $\{a\}$ for sort s and $\varnothing$ for sort s' with operations being the identities. Restriction to the ONE component forgets about the identity on $\{a\}$ for sort s. But the meaning of the operator (name) f in TWO is given as the union of the function with the same name, i.e., the identity on $\{a\}$, which is not preserved as the meaning of f if ONE is the identity on the $\varnothing$.

This observation has repercussions with regard to correctness of parameter passing: Assume that ONE is the parameter part of

pres THREE **is**
sorts $\$s, \$s', s''$
ops $\$a: \to s, a: \to s''$
$\$f: s' \to s', f: s'' \to s''$

Parameter passing (i.e., the application of THREE to TWO) yields

pres FOUR **is**
sorts s, s', s''
ops $a: \to s, a: \to s''$
$f: s \to s, f: s' \to s', f: s'' \to s''$
eqns $f(a) = a$

The free THREE-algebra over $(T_{\mathrm{TWO}})_{\mathrm{ONE}}$ has carriers $\{f^n(a)\}$ for sort s and $\varnothing$ for sort s'' while the restriction of the initial FOUR-algebra (which is free over T_{TWO}) to its THREE-component has carriers $\{a\}$ for sort s''.

A comparison with the equivalent conditional equational specifications reveals the anomaly of the example. If we translate the subsorting to conditional equational specifications (roughly according to 2.1 and 2.2), we obtain

pres ONE′ **is**
sorts s, s'
ops $a_s: \to s$
$f_{s'}: s' \to s'$

$\subseteq$

pres THREE′ **is**
sorts $s, s', s'', s \cap s', s \cap s''$
ops $a_s: \to s, a_{s''}: \to s''$
$a_{s \cap s''}: s \cap s''$
$f_{s'}: s' \to s', f_{s''}: s'' \to s''$
$f_{s' \cap s''}: s' \cap s'' \to s' \cap s''$
$m_1: s \cap s'' \to s$
$m_2: s \cap s'' \to s''$
$m_3: s' \cap s'' \to s'$
$m_4: s' \cap s'' \to s''$

eqns $m_1(a_{s\cap s''}) = a_s$
$m_2(a_{s\cap s''}) = a_{s''}$
$m_3(f_{s'\cap s''}(x)) = f_{s'}(m_3(x))$
$m_4(f_{s'\cap s''}(y)) = f_{s''}(m_4(y))$
$m_1(z) = m_1(z') \Rightarrow z = z'$
+"other cond. eq. to state that
m_2, m_3, m_4 are injective"

and the actual parameter

pres TWO′ **is**
sorts $s, s', s \cap s'$
ops $a_s: \to s$
$f_s: s \to s, f_{s'}: s' \to s'$
$f_{s\cap s'}: s \cap s' \to s \cap s'$
$m_5: s \cap s' \to s$
$m_6: s \cap s' \to s'$
eqns $f_s(a_s) = a_s$
$m_5(f_{s\cap s'}(x)) = f_s(m_5(x))$
$m_6(f_{s\cap s'}(y)) = f_{s'}(m_6(y))$
+"cond. eq. to state that
m_5, m_6 are injective"

the pushout of which is the union of TWO′ and THREE′. The identification of "$f_{s''}(a_{s''}) = a_{s''}$" in FOUR cannot takes place in FOUR′.

We conclude that parameter passing for presentations with subsorts behaves differently compared to parameter passing for the equivalent conditional equational specifications. This phenomenon is inherent as order-sorting generates identities on overloaded operations which are not automatically generated in conditional equational specifications.

We encounter the same kind of problem if order-sorted presentations à la [19] are considered (compare Section 2.1):

pres	PADHOC1 **is**		**pres**	ADHOC1 **is**
sorts	s, s'	$\subseteq$	**sorts**	s, s'
ops	$f: s' \to s'$		**ops**	$f: s \to s$
				$f: s' \to s'$
	↓			↓
pres	SUBSORT1 **is**		**pres**	OVERLOAD1 **is**
sorts	$s < s'$	$\subseteq$	**sorts**	$s < s'$
ops	$a: \to s$		**ops**	$a: \to s$
	$f: s' \to s'$			$f: s \to s$
				$f: s' \to s'$
eqns	$f(a) = a$		**eqns**	$f(a) = a$

This is a pushout of presentations although the result is counterintuitive in that the parametrized specification states "ad hoc polymorphism" of the overloaded operators but parameter passing converts "ad hoc polymorphism" to "parametric

polymorphism" (by the way, passing compatibility does not hold). Again, the pushout of the corresponding conditional equational specifications

pres	PADHOC1′ **is**		**pres**	ADHOC1′ **is**
sorts	s, s'	$\subseteq$	**sorts**	s, s'
ops	$f: s' \to s'$		**ops**	$f: s \to s$
				$f': s' \to s'$
	$\downarrow$			$\downarrow$
pres	SUBSORT1′ **is**		**pres**	OVERLOAD1′ **is**
sorts	s, s'	$\subseteq$	**sorts**	s, s'
ops	$a: \to s$		**ops**	$a: \to s$
	$f: s' \to s'$			$f: s \to s$
	$m: s \to s'$			$f': s' \to s'$
eqns	$f(a) = a$			$m: s \to s'$
	$m(x) = m(y) \Rightarrow x = y$		**eqns**	$f(a) = a$
				$m(x) = m(y) \Rightarrow x = y$

behaves differently in that the equation "$m(f(x)) = f'(m(x))$" does not hold, which is implicit in OVERLOAD1.

Remark. Regularity of presentations (see Section 2.1) is not preserved:

pres	PADHOC2 **is**		**pres**	ADHOC2 **is**
sorts	s, s'	$\subseteq$	**sorts**	s, s'
ops	$f: s \to s$		**ops**	$f: s \to s$
				$f: s' \to s'$
	$\downarrow$			$\downarrow$
pres	SUBSORT2 **is**		**pres**	OVERLOAD2 **is**
sorts	$s'' \leqslant s,\ s'' \leqslant s'$	$\subseteq$	**sorts**	$s'' \leqslant s,\ s'' \leqslant s'$
ops	$f: s \to s$		**ops**	$f: s \to s$
				$f: s' \to s'$

Thus pushouts of presentations are not appropriate to model parameter passing for order-sorted algebras as in [19]. There seem to be difficulties to modify the setting so that regularity is preserved by parameter passing.

Obviously, there are two divergent aims: on the one hand, we would like to see subsorting as a special case of conditional equational specifications; on the other hand, we would like to retain the free constructions as given in the second section which inherently demand uniform renaming of the operators. We have to narrow our view of presentations in order to reconcile these aims. The first example in this section demonstrates that the implicit definition of "meaning" of an operator (name) is technically problematic, and, I claim, has been somewhat counterintuitive from the very beginning. The philosophy has always been that an operator is only determined by its name and that the type of information only refers to restricted instances of such an operator, the paradigm being that "the same operator applied to the same data should yield the same result." However, we have avoided providing a syntactical representation of the denotation of the "meaning" of an operator. This can be made by assuming the existence of "maximal" (w.r.t. coercion) instan-

ces of operators. Presentations will be called "specifications" if every operator has a maximal instance.

Parameter passing should then preserve the "meaning" of operators. Hence one would expect the formal parameter to be a specification and parameter passing to preserve the meaning of operators in that maximal instances of operators are mapped to maximal instances. These conditions are not yet sufficient:

EXAMPLE.

spec	PARAM **is**		**spec**	EXAM **is**
sorts	s		**sorts**	$s < s'$
ops	$a, b: \to s$	$\subseteq$	**ops**	$a, b: \to s$
	$f, g: s \to s$			$f, g: s \to s$
				$c: \to s'$
				$f, g: s' \to s'$
			eqns	$f(c) = a \quad g(c) = b$
	↓			↓
spec	ASPEC **is**		**spec**	RSPEC **is**
sorts	s		**sorts**	$s < s'$
ops	$a, b: \to s$	$\subseteq$	**ops**	$a, b: \to s$
	$f: s \to s$			$f: s \to s$
				$c: \to s'$
				$f: s' \to s'$
			eqns	$f(c) = a \quad f(c) = b$

As $a \neq b$ in the initial ASPEC-algebra but $a = b$ in the initial RSPEC-algebra parameter protection is not guaranteed. If we consider the corresponding conditional equational specifications

spec	PARAM′ **is**		**spec**	EXAM′ **is**
sorts	s		**sorts**	s, s'
ops	$a, b: \to s$	$\subseteq$	**ops**	$a, b: \to s$
	$f_s, g_s: s \to s$			$f_s, g_s: s \to s$
				$c: \to s'$
				$f_{s'}, g_{s'}: s' \to s'$
				$m: s \to s'$
			eqns	$f_{s'}(c) = m(a)$
				$g_{s'}(c) = m(b)$
				$f_{s'}(m(x)) = m(f_s(x))$
				$g_{s'}(m(x)) = m(g_s(x))$
				$m(x) = m(y) \Rightarrow x = y$
	↓			↓
spec	ASPEC′ **is**		**spec**	RSPEC′ **is**
sorts	s		**sorts**	s, s'
ops	$a, b: \to s$	$\subseteq$	**ops**	$a, b: \to s$
	$f: s \to s$			$f: s \to s$
				$c: \to s'$
				$f_{s'}, g_{s'}: s' \to s'$
				$m: s \to s'$

$$\begin{aligned}\textbf{eqns}\quad & f_{s'}(c) = m(a)\\ & g_{s'}(c) = m(b)\\ & f_{s'}(m(x)) = m(f(x))\\ & g_{s'}(m(x)) = m(g(x))\\ & m(x) = m(y) \Rightarrow x = y\end{aligned}$$

the identifications $f_{s'}(c) = g_{s'}(c)$ and $a = b$ do not take place.

We conclude that identification of operator names causes difficulties if carried over to operators which are not in the parameter component. Therefore I have argued in [23, 24] that parameter passing has to be restricted to satisfy certain safeness conditions. Safeness of parameter passing, however, does not guarantee that the parameter passing for order-sorted specifications mirrors the parameter passing for the corresponding conditional equational specifications. This can only be achieved if we strictly separate between operator names which occur in the formal parameter and those in the body of a parametrized specification. This requirement is justified pragmatically as there are few natural examples where the same operator names may be used in the parameter and body of a specification.

Existence of maximal operators is rather a drastic assumption. One may use the weaker condition that operators with the same name are "related by coercion" in that they are in the transitive closure of the coercion relation. But the weaker notion is more awkward to handle and less natural if we interpret operators with the same name as instances of the same operator. One should note that ad hoc polymorphism is excluded in any case. The counter examples above suggest that it may be difficult to develop a theory of parametrization for order-sorted specifications which allow ad hoc polymorphism.

3.3. *The Syntax of Parameter Passing*

The parametrization mechanism is defined with regard to a category of specifications.

DEFINITION. A presentation (S, Σ, E) is called a *specification* if for all operators $\sigma \in \Sigma$ there exists a *maximal* instance $\sigma: \sigma^- \to \sigma^+$, i.e., $\sigma: w \to s \in \Sigma$ implies $w \leqslant \sigma^-$ and $s \leqslant \sigma^+$. σ^- and σ^+ are called the *arity* (resp. *coarity*) of σ.

A *morphism of specifications* h: SPEC $\to$ SPEC′ is a presentation morphism which preserve maximality, i.e., $h(\sigma: \sigma^- \to \sigma^+) = h(\sigma): h(\sigma^-) \to h(\sigma^+)$.

These data define a category SPEC of specification which is a subcategory of PRES.

We proceed now along the lines of [8], using pushouts in the category of specifications to model parameter passing.

DEFINITION. A *parametrized specification* SPEC $= (S, \Sigma, E)$ is a specification with a *formal parameter* PSPEC $= (PS, P\Sigma, PE)$ s.t.

- PSPEC is a subspecification of SPEC,
- *parameter completeness* holds, i.e., if $s \in S$, $s' \in PS$, and $s \leqslant s'$ then $s \in PS$.

A *parameter passing* (*morphism*) is a specification morphism h: PSPEC $\to$ ASPEC. The parametrized specification ASPEC is called *actual parameter*.

Remark. Parameter completeness is a natural condition which eases the technical development. It does not seem to be a restriction of any importance as one might replace $s \leqslant s'$, where $s \in S$, $s' \in PS$ by a monomorphic operator $m: s \to s'$ with suitable axioms.

Notation. Throughout the rest of the paper we consider a fixed parametrized specification SPEC with formal parameter PSPEC and an actual parameter ASPEC such that

$$\text{PSPEC} = (PS, P\Sigma, PE) \subseteq \text{SPEC} = (S, \Sigma, E)$$

$$\text{PASPEC} = (PAS, PA\Sigma, PAE) \subseteq \text{ASPEC} = (AS, A\Sigma, AE),$$

where $S = (PS + BS, \leqslant)$, $\Sigma = P\Sigma + B\Sigma$ and $E = PE + BE$ ("B" stands for "body"). We assume without restriction of generality that $BS \cap AS = \varnothing$ and $B\Sigma + A\Sigma = \varnothing$. Note that $P\Sigma \cap B\Sigma = \varnothing$ as PSPEC is a subspecification of SPEC.

The category of specifications has pushouts. We only discuss pushouts of the specific structure in order to avoid unnecessary complications.

DEFINITION. Let RSPEC $= (RS, R\Sigma, RE)$ be defined by

$$RS = (AS + BS), \qquad R\Sigma = A\Sigma + h'(B\Sigma), \qquad RE = AE + h'(BE)$$

where h': SPEC $\to$ RSPEC is given by

$$h'(s) = \text{if } s \in PS \quad \text{then } h(s) \text{ else } s$$

$$h'_{w,s}(\sigma) = \text{if } \sigma \in P\Sigma \quad \text{then } h(\sigma) \text{ else } \sigma$$

and where the partial order on RS is the transitive closure of $s_1 \leqslant s_2$ where

$s_1 \leqslant s_2$ if
- $s_1 \leqslant s_2$ in AS, or
- $s_1 = h'(s'_1)$ and $s_2 = h'(s'_2)$ for some $s'_1 \leqslant s'_2$ in S.

3.1. PROPOSITION.

$$\begin{array}{ccc} \text{PSPEC} & \underset{p}{\subseteq} & \text{SPEC} \\ {\scriptstyle h}\downarrow & & \downarrow{\scriptstyle h'} \\ \text{ASPEC} & \underset{p'}{\subseteq} & \text{RSPEC} \end{array}$$

is a pushout diagram in **SPEC** *and* PASPEC $\subseteq$ RSPEC *is parameter complete.*

Proof. We check that $(AS + BS, \leqslant)$ is well defined: The relation is reflexive and transitive by definition. Assume that $s_1 \leqslant s_2$ and $s_2 \leqslant s_1$.

(a) If $s_1, s_2 \in BS$ then $s_1 = s_2$ as h' is the identity on BS.

(b) Because of monotonicity of h the order relation on the AS-component of RS is that of AS in ASPEC. Thus $s_1 = s_2$ for $s_1, s_2 \in AS$.

(c) if $s_1 \in AS$ and $s_2 \in BS$ then there exist $s_1', s_1'' \in PS$, $s_2', s_2'' \in BS$ such that $h'(s_1') = h'(s_1'') = s_1$ and $h'(s_2') = h'(s_2'') = s_2$ and $s_1' \leqslant s_2'$, $s_2'' \leqslant s_1''$. But $s_2'' \leqslant s_1''$ cannot hold because of parameter completeness (in fact, this motivates parameter completeness; otherwise one needs to factorize by antisymmetry which rather obscures the subsort ordering on RS).

PASPEC $\subseteq$ RSPEC is parametrized specification: Maximal operators are the maximal operators in $A\Sigma$ and $h'(B\Sigma)$ due to the disjointness of operator sets. For parameter completeness, let $s \in RS$ and $s' \in AS$ with $s \leqslant s'$ in RS. If $s \in AS$ then $s \in PAS$ by parameter completeness of ASPEC. Otherwise, $s = h'(s'')$ with $s'' \in BS$. Then there exists a $s''' \in PS$ such that $s'' \leqslant s'''$ and $h(s''') \leqslant s$ by definition of the partial order on RS. Parameter completeness of SPEC implies that $s'' \in PS$ and $s \in AS$ which is a reduction to the first case.

The sort component of h' is monotone by definition, and the overloading condition of presentation morphisms are satisfied as the definition of h' only depends on operator names. Maximal operators are preserved as $A\Sigma \cap B\Sigma = \varnothing$ and $P\Sigma \cap B\Sigma = \varnothing$.

Now let

$$\begin{array}{ccc} \text{PSPEC} & \underset{p}{\subseteq} & \text{SPEC} \\ {\scriptstyle h}\downarrow & & \downarrow{\scriptstyle g} \\ \text{ASPEC} & \xrightarrow[f]{} & \text{SPEC}' \end{array}$$

be a commutative diagram in **SPEC** with $\text{SPEC}' = (S', \Sigma', E')$.

The underlying set diagram of the sort component induces a mapping

$$k: AS + BS \to S', \qquad \begin{array}{ll} k(s) = f(s) & \text{if } s \in AS \\ k(s) = g(s) & \text{if } s \in BS. \end{array}$$

We have $k(p'(s)) = k(s) = f(s)$ for $s \in AS$ and $k(h'(s)) = k(h(s)) = f(h(s)) = g(p(s)) = g(s)$ if $s \in PS$ and $k(h'(s)) = k(s) = g(s)$ if $s \in BS$. Clearly, k is the unique such mapping. Let $s_1 \leqslant s_2$ in RS. If $s_1, s_2 \in AS$ then $k(s_1) = f(s_1) \leqslant f(s_2) = k(s_2)$, and if $s_1, s_2 \in BS$ then $k(s_1) = g(s_1) \leqslant g(s_2) = k(s_2)$. If $s_1 \in PS$ and $s_2 \in BS$ then $k(h'(s_1)) = g(s_1) \leqslant g(s_2) = k(h'(s_2))$. Hence k is monotone.

On operators we define the mappings

$$k_{w,s}: R\Sigma_{w,s} \to \Sigma'_{k(w),k(s)}, \qquad \begin{array}{ll} k_{w,s}(\sigma) = f_{w,s}(\sigma) & \text{if } \sigma: w \to s \in A\Sigma \\ k_{w,s}(\sigma) = g_{w',s'}(\sigma) & \text{if } \sigma: w' \to s' \in B\Sigma \text{ and} \\ & h'(\sigma: w' \to s') = \sigma: w \to s \end{array}$$

(Note that h' does not rename operators in $B\Sigma$ as $P\Sigma \cap B\Sigma = \emptyset$.) This is well defined as $g(s) = k(h'(s))$ for $s \in PS + BS$.

We check that k: RSPEC $\to$ SPEC′ is the unique presentation morphism such that $k \circ p' = f$ and $k \circ h' = g$: Let $\sigma: w_1 \to s_1$, $\sigma: w_2 \to s_2 \in R\Sigma$. If $\sigma: w_1 \to s_1$, $\sigma: w_2 \to s_2 \in A\Sigma$ then $k_{w_1,s_1}(\sigma) = f_{w_1,s_1}(\sigma) = f_{w_2,s_2}(\sigma) = k_{w_2,s_2}(\sigma)$. If $\sigma: w_1' \to s_1'$, $\sigma: w_2'' \to s_2'' \in B\Sigma$ with $h'(\sigma: w_1' \to s_1') = \sigma: w_1 \to s_1$, $h'(\sigma: w_2'' \to s_2'') = \sigma: w_2 \to s_2$ then $k_{w_1,s_1}(\sigma) = g_{w_1',s_1'}(\sigma) = g_{w_2',s_2'}(\sigma) = k_{w_2,s_2}(\sigma)$. Hence k is a presentation morphism. Maximal operators are preserved because of the various disjointness conditions.

Let $\sigma: w \to s \in A\Sigma$. Then $k_{w,s}(p_{w,s}(\sigma)) = k_{w,s}(\sigma) = f_{w,s}(\sigma)$. If $\sigma: w \to s \in P\Sigma$ then $k_{h'(w),h'(s)}(h'_{w,s}(\sigma)) = k_{h(w),h(s)}(h_{w,s}(\sigma)) = f_{h(w),h(s)}(h_{w,s}(\sigma)) = g_{w,s}(\sigma)$, and $k_{h(w),h(s)}(h'_{w,s}(\sigma)) = g_{w,s}(\sigma)$ for $\sigma: w \to s \in B\Sigma$.

For unicity we observe that given a morphism k': RSPEC $\to$ SPEC′ with $k' \circ p' = f$ and $k' \circ h' = g$ it holds that $k'_{w,s}(\sigma) = k'_{w,s}(p'_{w,s}(\sigma)) = f_{w,s}(\sigma) = k_{w,s}(\sigma)$ for $\sigma: w \to s \in A\Sigma$ and $k'_{h'(w),h'(s)}(h'_{w,s}(\sigma) = g_{w,s}(\sigma) = k_{h'(w),h'(s)}(\sigma)$ for $\sigma: w \to s \in B\Sigma$. ∎

DEFINITION. The *syntax* of parameter passing is given by the diagram

$$\begin{array}{ccc} \text{PSPEC} & \underset{p}{\subseteq} & \text{SPEC} \\ {\scriptstyle h}\downarrow & & \downarrow{\scriptstyle h'} \\ \text{PASPEC} & \underset{p_A}{\subseteq} \text{ASPEC} \underset{p'}{\subseteq} & \text{RSPEC} \end{array}$$

where the square is a pushout in **SPEC**.

3.4. *The Semantics of Parameter Passing*

As for standard many-sorted specifications parameter data should be protected [8].

DEFINITION. A *parametrized data type* is a parametrized specification s.t. *parameter protection* holds, i.e., PSPEC $\subseteq$ SPEC is persistent.

A presentation embedding PRES $\subseteq$ PRES′ is *consistent* if the units of the adjunctions $A\eta$: $A \to (T_{\text{PRES}'}(A))_{\text{PRES}}$ of the corresponding adjunction (2.8) are injective. If the units are isomorphisms (identities) the embedding is called (*strongly*) *persistent.*

The definition somewhat contradicts the claim made in Section 3.1 that "sufficient completeness" has been abandoned, i.e., that persistency should be replaced by consistency. But "sufficient completeness" must be reinterpreted in the presence of subsorts. Let $s \leqslant s'$ where $s \in PS$ and $s' \in S$. Parameter data of sort s are as well data of sort s' but there may be other data of type s'. s' may be seen as a "parameter sort" in the sense that it is partly determined by data generated by an actual parameter though it is not necessarily part of the formal parameter, e.g., *data*+ in the array specification

spec ARRAY
BOOL **with**
sorts $\$data < data+$, $\$index$, $array < array+$
ops $\$eq$: *index index* → *bool*
new: → *array*
get: *array index* → *data*
update: *array index data* → *array*
update: *array*+ *index data*+ → *array*+
if_then_else_: *bool array*+ *array*+ → *array*+
if_then_else_: *bool data*+ *data*+ → *data*+
derror: → *data*+
aerror: → *array*+
...

This more general notion of a "parametrized sort" induces a new interpretation of "sufficient completeness":

DEFINITION. A parametrized specification is *sufficient complete* if $T_{\mathrm{SPEC}}(A)_{s'} = \bigcup \{T_{\mathrm{SPEC}}(A)_s \mid s \in PS,\ s \leqslant s'\}$ for all *parametrized sorts* $s' \in S$, i.e., sorts $s' \in S$ s.t. there exists a $s \in PS$ with $s \leqslant s'$.

ARRAY is an example of a parametrized specification which is not sufficiently complete. From this point of view, a parametrized specification is only "consistent" in that parameter data are not identified. This observation suggests a syntactical criterium to guard the parameter part against data imported from the body of a specification.

DEFINITION. A presentation SPEC with a parameter PSPEC is *strongly parameter complete* if $\sigma: w \to s \in \Sigma$ and $s \in PS$ implies that $\sigma: w \to s \in P\Sigma$.

Typically, this property is satisfied by specifications where error elements are added to parameter data. Strong parameter completeness is helpful because of the

3.2. PROPOSITION. *Parameter protection holds if a parametrized specification is strongly parameter complete and consistent.*

The proposition is a consequence of

3.3. LEMMA. $A = (T_{\mathrm{SIG}}(A))_{\mathrm{PSIG}}$ *if* $\mathrm{PSIG} \subseteq \mathrm{SIG}$ *is strongly parameter complete, A being a PSIG-algebra.*

Proof. By strong parameter completeness there is no term of the form $\sigma(\mathbf{t})$: $s \in T_{\mathrm{SIG}}(A)$ with $s \in PS$ (compare 2.12). ∎

The following correctness criteria formalize the intuitions provided in Section 3.1 (compare [8]).

DEFINITION. The *semantics* of parameter passing

$$\begin{array}{ccc} \mathrm{PSPEC} & \underset{p}{\subseteq} & \mathrm{SPEC} \\ {\scriptstyle h}\downarrow & & \downarrow{\scriptstyle h'} \\ \mathrm{PASPEC} & \underset{p_A}{\subseteq} \mathrm{ASPEC} \underset{p'}{\subseteq} & \mathrm{RSPEC} \end{array}$$

is given by the free functor

$$F_{p'} \circ F_{p_A} : \mathbf{OSAlg}_{\mathrm{PASPEC}} \to \mathbf{OSAlg}_{\mathrm{RSPEC}}.$$

Parameter passing is *correct* if

- RSPEC with formal parameter PASPEC is a parametrized data type, and

if

- *passing compatibility* holds, i.e., $F_p \circ V_h \circ F_{p_A} \approx V_{h'} \circ F_{p'} \circ F_{p_A}$.

3.4. MAIN THEOREM. *Parameter passing is correct if* SPEC *is a parametrized data type.*

The proof is split into several lemmas below.

Let A be a ASPEC-algebra. Then we have the commutative diagram

$$\begin{array}{ccccc} & \overset{\eta}{\nearrow} & (T_{\mathrm{SIG}}(A_h))_{\mathrm{PSIG}} & & T_{\mathrm{SIG}}(A_h) \xrightarrow{\pi} T_{\mathrm{SPEC}}(A_h) \\ A & & \downarrow{\scriptstyle f} & & \downarrow{\scriptstyle f} \qquad \downarrow{\scriptstyle g} \\ & \underset{\eta'_h}{\searrow} & (T_{\mathrm{RSIG}}(A)_h)_{\mathrm{PSIG}} & & T_{\mathrm{RSIG}}(A)_h \xrightarrow[\pi'_h]{} T_{\mathrm{RSPEC}}(A)_h \end{array}$$

in $\mathbf{OSAlg}_{\mathrm{PSIG}}$ in $\mathbf{OSAlg}_{\mathrm{SIG}}$

η and η' are the unit of the adjunctions $\mathbf{OSAlg}_{\mathrm{SIG}} \to \mathbf{OSAlg}_{\mathrm{PSIG}}$ (resp. $\mathbf{OSAlg}_{\mathrm{RSIG}} \to \mathbf{OSAlg}_{\mathrm{ASIG}}$). π and π' are the surjections induced by factorization. f is uniquely generated by freeness of $T_{\mathrm{SIG}}(A_h)$ (compare 2.12) and g by freeness of $T_{\mathrm{SPEC}}(A_h)$. Observe that $\pi \circ \eta$ and $\pi' \circ \eta$ are the units of the adjunctions $\mathbf{OSAlg}_{\mathrm{SPEC}} \to \mathbf{OSAlg}_{\mathrm{PSPEC}}$ (resp. $\mathbf{OSAlg}_{\mathrm{RSPEC}} \to \mathbf{OSAlg}_{\mathrm{ASPEC}}$). We use $\__h : \mathbf{OSAlg}_{\mathrm{ASIG}} \to \mathbf{OSAlg}_{\mathrm{PSIG}}$ and $\__h : \mathbf{OSAlg}_{\mathrm{RSIG}} \to \mathbf{OSAlg}_{\mathrm{SIG}}$ ambiguously.

Remark. Subsequently we use induction on $T_{\mathrm{SIG}}(\hat{A})$ in proofs over $T_{\mathrm{SIG}}(A)$ ($T_{\mathrm{SIG}}(A)$ denotes the free SIG-algebra, $T_{\mathrm{SIG}}(\hat{A})$ the free term algebra over the underlying set of generators). This is justified as $\__A : T_{\mathrm{SIG}}(\hat{A}) \to T_{\mathrm{SIG}}(A)$ is surjective (2.12).

3.5. LEMMA. (i) *If* $s' \leqslant h'(s)$ *(resp.* $h(s) \leqslant s'$*) for* $s \in S$ *then there exists a* $s'' \in PS$ *such that* $s' \leqslant h'(s'')$ *and* $s'' \leqslant s$ *(resp.* $h'(s'') \leqslant s'$ *and* $s \leqslant s''$*).*

(ii) *If* $a: s' \in T_{\mathrm{RSIG}}(\hat{A})$ *with* $a \in \hat{A}$ *and* $s' \in RS$ *then there exists a* $s \in AS$ *such that* $a: s \in \hat{A}$ *and* $s \leqslant s'$.

Proof. (i) The non-trivial case is $s \in BS$. Order of RS is inherited from AS or from S via h'. As $h'(BS) = BS$ and $BS \cap AS = \varnothing$ we have that $s'' \leqslant h'(s)$ only holds if there exists a $s'' \in PS$ with the required properties.

(ii) Constructing $T_{\mathrm{RSIG}}(\hat{A})$ the AS-indexed set $\hat{A}$ is expanded to a RS-indexed set $\hat{A}$. This is the only generation of elements of $\hat{A}$ in $T_{\mathrm{RSIG}}(\hat{A})$. ▮

3.6. LEMMA. *Let* $t: s \in T_{\mathrm{SIG}}(A_h)$. *Then*

(a) $f_s(t) = a$ *iff* $t = a$ *for all* $a \in A$,

(b) $f_s(t) = h(\sigma)(f_w(\mathbf{t}))$ *if* $t = \sigma(\mathbf{t})$ *for* $\sigma: w \to s \in \Sigma$ *and* $\mathbf{t}: w \in T_{\mathrm{SIG}}(A_h)$, *and*

(c) $t = \sigma(\mathbf{t})$ *if* $f_s(t) = \sigma'(\mathbf{t}')$ *for some* $\sigma: w \to s \in \Sigma$ *and* $\mathbf{t}: w \in T_{\mathrm{SIG}}(A_h)$ *s.t.* $h(\sigma) = \sigma'$ *and* $f_w(\mathbf{t}) = \mathbf{t}'$.

Proof. The diagram

$$\begin{array}{ccc} T_{\mathrm{SIG}}(\hat{A}) & \xrightarrow{-_{A_h}} & T_{\mathrm{SIG}}(A) \\ \Big\downarrow{\scriptstyle f'} & & \Big\downarrow{\scriptstyle f} \\ T_{\mathrm{RSIG}}(\hat{A})_h & \xrightarrow[(-_A)_h]{} & T_{\mathrm{RSIG}}(A)_h \end{array}$$

commutes due to universal properties.

The statement clearly holds if we replace f by f'. We use induction over $t \in T_{\mathrm{SIG}}(\hat{A})$:

$t = a: s \in A$. Then $a_{A_h} = a$ and $a_A = a$.

$t = \sigma(\mathbf{t}): s$. Either $\sigma \in P\Sigma$ then $(\sigma^{w,s}(\mathbf{t}))_{A_h} = \sigma^{w,s}_{A_h}(\mathbf{t}_A) = \sigma^{h(w),h(s)}_A(\mathbf{t}_A) \in A$, and $h'(\sigma)(f'(\mathbf{t}))_A = h'(\sigma)^{h(w),h(s)}_A(f'(\mathbf{t})_A) = h'(\sigma)^{h(w),h(s)}_A(\mathbf{t}_A)$, where $\mathbf{t}_A \in A$ with $f'(\mathbf{t}_A) = \mathbf{t}_A$ being the inductive assumption. We can assume without restriction of generality that $w = \sigma^-$ and $s = \sigma^+$. Then $h'(\sigma: \sigma^- \to \sigma^+) = h'(\sigma): h'(\sigma)^- \to h'(\sigma)^+$ as parameter passing preserves maximal operators. Hence the argument holds for the other direction.

Otherwise $(\sigma^{w,s}(\mathbf{t}))_{A_h} = \sigma(\mathbf{t}_{A_h})$. Then $h'(\sigma)(f'(\mathbf{t}))_A = h'(\sigma)(f'(\mathbf{t})_A)$ and the inductive assumption can be used. ▮

3.7. LEMMA. *Let* $t': s' \in T_{\mathrm{RSIG}}(A)$. *Then*

(a) *there exists a* $t: s \in T_{\mathrm{SIG}}(A_h)$ *s.t.* $f(t) \approx t'$ *if* $s' = h'(s)$ *for some* $s \in S$, *and*

(b) $t' \approx a: s' \in A$ *if* $s' \in AS$.

Proof. Let $t': h'(s) \in T_{\mathrm{RSIG}}(A)$. We use induction over terms in $T_{\mathrm{RSIG}}(\hat{A})$:

(i) $t' \in \hat{A}$: $a: h'(s) \in T_{\mathrm{RSIG}}(\hat{A})$ implies existence of an $s' \in AS$ s.t. $a: s' \in \hat{A}$ and

$s' \leqslant h'(s)$ (by 3.5(ii)) and of a $s'' \in PS$ s.t. $s' \leqslant s$ (3.5(i)). Hence $a: s \in \hat{A}_h$ and $f_s(a) = f_{s'}(a) = a$.

(ii) $t' = \sigma(\mathbf{t}')$ with $\sigma: s'_1 \cdots s'_n \to s' \in A\Sigma$, $\mathbf{t}' \in T_{\mathrm{RSIG}}(\hat{A})$: Then $t'_j \approx a_j \in \hat{A}$ by inductive assumption. Congruence ensures that $\sigma(t_1, ..., t_n) \approx \sigma(a_1, ..., a_n) \approx \sigma_A^{s_1 \cdots s_n, s}(a_1, ..., a_n) \in \hat{A}$. This is a reduction to (i).

(iii) $t' = \sigma'(\mathbf{t}')$ with $\sigma': s'_1 \cdots s'_n \to s' \in h'(B\Sigma)$, $\mathbf{t}' \in T_{\mathrm{RSIG}}(\hat{A})$: There exists some $\mathbf{t}: s_1 \cdots s_n \in T_{\mathrm{SIG}}(\hat{A}_h)$ such that $f(\mathbf{t}) \approx \mathbf{t}'$ by inductive assumption as all s_i are in the image of h'. Then $\sigma(\mathbf{t}): s \in T_{\mathrm{SIG}}(\hat{A}_h)$ and $f(\sigma(\mathbf{t})) = h'(\sigma)(f(\mathbf{t})) = \sigma'(\mathbf{t}) \approx \sigma'(\mathbf{t}')$, where $h'(\sigma: s_1 \cdots s_n \to s) = \sigma': s'_1 \cdots s'_n \to s'$. If $s \in PS$ then there exists an $a: s \in A_h$ s.t. $\sigma(\mathbf{t}) \approx a$ w.r.t. $T_{\mathrm{SPEC}}(A_h)$. Then $\sigma'(\mathbf{t}') \approx f(\sigma(\mathbf{t})) \approx f(a) = a$ w.r.t. $T_{\mathrm{RSPEC}}(A)$. ∎

3.8. COROLLARY. (i) $g: T_{\mathrm{SPEC}}(A_h) \to T_{\mathrm{RSPEC}}(A)_h$ *is surjective.*

(ii) *The unit of the adjunction induced by* ASPEC ⊆ RSPEC *is surjective.*

Proof. (i) The lemma states that $\pi' \circ f: T_{\mathrm{SIG}}(A_h) \to T_{\mathrm{RSPEC}}(A)_h$ is surjective. Then $g: T_{\mathrm{SPEC}}(A_h) \to T_{\mathrm{RSPEC}}(A)_h$ is surjective.

(ii) is a reformulation of 3.7(b). ∎

3.9. LEMMA. $t \approx t'$ (mod $T_{\mathrm{SPEC}}(A_h)$) *if* $f_s(t) = f_{s'}(t')$ *for all terms* $t: s$, $t': s' \in T_{\mathrm{SIG}}(A_h)$.

Proof. We use induction on $T_{\mathrm{SIG}}(\hat{A})$:

$t = a \in \hat{A}$. Then $f_s(t) = a = f_{s'}(t')$ and $t' = a$ by 3.6.

$t = \sigma(\mathbf{t})$. Then $f_s(t) = h'(\sigma)(f_w(\mathbf{t}))$ and there exist $\sigma': w' \to s' \in P\Sigma$ and $\mathbf{t}': w' \in T_{\mathrm{SIG}}(A)$ such that $t = \sigma'(\mathbf{t}')$ and $f_{w'}(\mathbf{t}') = f_w(\mathbf{t})$ by 3.6. Moreover, $\mathbf{t} \approx \mathbf{t}'$ by inductive assumption.

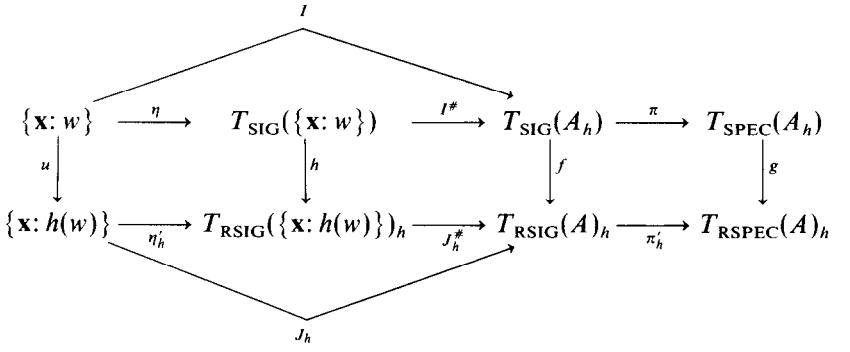
If $\sigma: w \to s \in P\Sigma$ then $\mathbf{t} \approx \mathbf{a}: w \in A$, $\mathbf{t}' \approx \mathbf{a}': w' \in A$ and $\mathbf{a} = \mathbf{a}'$ by parameter protection. We compute $\sigma(\mathbf{t}) \approx \sigma(\mathbf{a}) \approx \sigma_A^{w,s}(\mathbf{a}) = h(\sigma)_A^{h(w),h(s)}(\mathbf{a}) = \sigma_A'^{w',s'}(\mathbf{a}) \approx \sigma'(\mathbf{a}) \approx \sigma'(\mathbf{t}')$. Otherwise the inductive assumption can be used straightforwardly as $\sigma = \sigma'$. ∎

Remark. The lemma makes essential use of the fact that PSPEC is a subspecification of SPEC. Otherwise terms of the form $\sigma(\cdots\sigma'(\cdots)\cdots)$, where $\sigma \in P\Sigma$ and $\sigma' \in B\Sigma$ are not necessarily equivalent to elements of A (compare EXAM in Section 3.2).

3.10. PROPOSITION. *Passing compatibility holds for parametrized data types.*

Proof. We check $T_{\mathrm{SPEC}}(A_h) \cong T_{\mathrm{RSPEC}}(A)_h$ for $A \in \mathbf{OSAlg}_{\mathrm{ASPEC}}$. Because of 3.8 this holds if $g: T_{\mathrm{SPEC}}(A_h) \to T_{\mathrm{RSPEC}}(A)_h$ is injective, i.e., $g_s(\pi_s(t)) = g_s(\pi_s(t'))$ implies $\pi_s(t) = \pi_s(t')$ for $t, t': s \in T_\Sigma(A_h)$. But $g_s(\pi_s(t)) = g_s(\pi_s(t'))$ iff $\pi'_{h(s)}(f_s(t)) = \pi'_{h(s)}(f_s(t'))$. Hence g is injective if $f_s(t) \approx f_s(t')$ w.r.t. $T_{\mathrm{RSPEC}}(A)_h$ implies $t \approx t'$ w.r.t. $T_{\mathrm{SPEC}}(A_h)$. This is established by inspection of cases:

(i) $f_s(t) = J^{\#}_{h(s)}(h_s(l))$, $f_s(t') = J^{\#}_{h(s)}(h_s(r))$ with $[\mathbf{x}: w]l =_s r \in BE$ and $J: \{\mathbf{x}: h(w)\} \to T_{\mathrm{RSIG}}(A)$. We claim that there exists a $I: \{\mathbf{x}: w\} \to T_{\mathrm{SIG}}(A_h)$ such that



commutes. Assume that $J_{h(s')}(x)$ is not in the image of f. Then $J_{h(s')}(x)$ is of the form $\sigma(\mathbf{t})$ as otherwise $a: h(s) \in A$. If x occurs in l (or r) then $J_{h(s')}(x)$ occurs as a subterm in $J^{\#}_{h(s)}(h_s(l))$ (by definition of $T_{\mathrm{RSIG}}(A)$). But $f_s(t) = J^{\#}_{h(s)}(h_s(l))$, hence all subterms must be in the image of f.

If x does not occur in l and r any term in $T_{\mathrm{SIG}}(A_h)$ may be chosen. A suitable term must exist because of 3.8. Then $I^{\#}_s(l) \approx I^{\#}_s(r)$ w.r.t. $T_{\mathrm{SPEC}}(A_h)$ and $f_s(I^{\#}_s(l)) = J^{\#}_{h(s)}(h_s(l))$ and $f_s(I^{\#}_s(r)) = J^{\#}_{h(s)}(h_s(r))$, and $t \approx I^{\#}_s(l) \approx I^{\#}_s(r) \approx t'$ by 3.9.

(ii) $f_s(t) \equiv J^{\#}_{s'}(l)$, $f_s(t') \equiv J^{\#}_{s'}(r)$ with $[\mathbf{x}: w]l =_{s'} r \in AE$ and $J: \{\mathbf{x}: w\} \to T_{\mathrm{RSIG}}(A)$. $J_w(\mathbf{x}) \in A$ without restriction of generality as $w \in AS^*$ (3.8). As A satisfies any equation of AE $f_s(t) = J^{\#}_{s'}(l) = J^{\#}_{s'}(r) = f_s(t')$. Then $t = t'$ by 3.6.

(iii) $f_s(t_1) = \sigma'(\mathbf{t}'_1)$, $f_s(t_2) = \sigma'(\mathbf{t}'_2)$ with $\sigma': w'_1 \to s'_1$, $\sigma': w'_2 \to s'_2 \in R\Sigma$, $\mathbf{t}'_1: w'_1$, $\mathbf{t}'_2: w'_2 \in T_{\mathrm{RSIG}}(A)$ and $\mathbf{t}'_1 \approx \mathbf{t}'_2$. $t_1 = \sigma_1(\mathbf{t}_1)$ and $t_2 = \sigma_2(\mathbf{t}_2)$ with $h(\sigma_i: w_i \to s_i) = \sigma': w'_i \to s'_i$, $\mathbf{t}_i: w_i \in T_{\mathrm{SIG}}(A_h)$ and $f_{w_i}(\mathbf{t}_i) = \mathbf{t}'_i$, $i = 1, 2$, by 3.6, and $\mathbf{t}_1 \approx \mathbf{t}_2$ by inductive assumption. We can assume without restriction of generality that the operators are maximal and that $w'_1 = w'_2$ and $s'_1 = s'_2$. There are two cases. Let $s_{1,j}$ and $s_{2,j}$ be the jth component of w_1 (resp. w_2):

(a) $s_{1,j} \neq s_{2,j}$. Then $s_{1,j}, s_{2,j} \in PS$ (according to the pushout construction sorts cannot be identified otherwise), $t'_{1,j} = t_{1,j} \approx t_{2,j} = t'_{2,j} \in A$ (using 3.6 and 3.7), and $t_{1,j} = t_{2,j}$ by parameter protection.

(b) $s_{1,j} = s_{2,j}$. Then $t_{1,j}, t_{2,j}: s_{1,j} \in T_{\mathrm{SIG}}(A_h)$. Hence $\mathbf{t}_1, \mathbf{t}_2: w_1 \in T_\Sigma(A_h)$, $\sigma_1(\mathbf{t}_1) \approx \sigma_1(\mathbf{t}_2)$ by congruence and $\sigma_1(\mathbf{t}_1) \approx \sigma_2(\mathbf{t}_2)$ by 3.9 observing that $f_{w_1}(\sigma_1(\mathbf{t}_2)) = f_{w_1}(\sigma_2(\mathbf{t}_2))$.

(iv) Transitivity follows as $f_s(t_1) \approx f_s(t_2) = f_{s'}(t_3) \approx f_{s'}(t_4)$ implies $t_1 \approx t_2 \approx t_3 \approx t_4$ using the inductive assumption that $t_1 \approx t_2$ and $t_3 \approx t_4$ and 3.9. Reflexivity and symmetry is straightforward. ∎

3.11. PROPOSITION. *Parameter protection is preserved for parametrized data types.*

Proof. We check that ASPEC $\subseteq$ RSPEC is consistent. Let $a: s'$, $a': s' \in A \subseteq T_{R\Sigma}(A)$ such that $a \neq a'$ in A and $a \approx a' \bmod T_{\mathrm{RSPEC}}(A)$. If $a \neq a'$ there exists an equation $[\mathbf{x}: w]l =_{s'} r \in RE$ such that (without restriction of generality) $a = I^{\#}_{s'}(l)$ and $I^{\#}_{s'}(r) = a'$ according to the definition of $\approx$ on $T_{\mathrm{RSIG}}(A)$. If $[\mathbf{x}: w]l =_{s'} r \in AE$ then $a = I^{\#}_{s'}(l) = I^{\#}_{s'}(r) = a'$. If $[\mathbf{x}: w]l =_{s'} r \in h(BE)$ then $s' = h(s)$ for some $s \in S$ and $a \approx a': s \bmod T_{\mathrm{SPEC}}(A_h)$ by 3.9 and $a = a'$ by consistency of PSPEC $\subseteq$ SPEC. ∎

This completes the proof of the main theorem. ∎

Inspection of the proof shows that the requirement if PSPEC being a subspecification has only be used for 3.9. Hence one may wonder if one cannot somewhat more exploit the order-sorted structure allowing for *weak parametrized specifications*, where PSPEC only is a subpresentation of SPEC (i.e., the condition $P\Sigma \cap B\Sigma = \varnothing$ is dropped). This has been the starting point of [23, 24]. The counterexamples above have demonstrated that correctness of parameter pasing cannot be expected if parameter passing is not restricted.

DEFINITION. Parameter passing is *safe* if it is

- *syntactically safe* if $h'(\sigma^-) = h'(\sigma'^-)$ and $h'(\sigma^+) = h'(\sigma'^+)$ in SPEC if $h(\sigma) = h(\sigma')$, and
- *semantically safe* if

$$h'(t) = h'(t') \qquad \text{implies that} \quad \vdash_{\mathrm{SPEC}(h)} [\mathbf{x}: w]\, t = t'$$

for $t, t' \in T_{\mathrm{SIG}}(\{\mathbf{x}: w\})$, $w \in S^*$, where

- $\mathrm{SPEC}(h) := \mathrm{SPEC} + (\varnothing, \varnothing, \{[\mathbf{x}: w]\, t = t' \mid w \in PS^*,\ t, t' \in T_{\mathrm{PSIG}}(\{\mathbf{x}: w\})$ and $\vdash_{\mathrm{ASPEC}} [\mathbf{x}: h(w)]\, h'(t) = h'(t')\})$

parametrized by h: PSPEC $\to$ ASPEC, and

- the mappings in

$$\begin{array}{ccc} \{\mathbf{x}: w\} & \xrightarrow{\eta} & T_{\mathrm{SIG}}(\{\mathbf{x}: w\}) \\ {\scriptstyle \eta}\downarrow & & \downarrow{\scriptstyle h'} \\ \{\mathbf{x}: h(w)\} & \xrightarrow[\eta]{} & T_{\mathrm{RSIG}}(\{\mathbf{x}: h(w)\}) \end{array}$$

are determined by adjunctions (η is uniformly used for units).

Remark. Syntactical safeness is a necessary condition as otherwise RSPEC is not a specification. The proof of 3.1 has only to be changed marginally.

3.12. LEMMA. *$T_{\mathrm{SPEC}}(A_h)$ satisfies $[\mathbf{x}: w]\, t = t'$ for all ASPEC-algebras A if $\vdash_{\mathrm{SPEC}(h)} [\mathbf{x}: w]\, t = t'$.*

Proof. We only have to check that $T_{\text{SPEC}}(T_h)$ satisfies equations $[\mathbf{x}: w]t = t'$ s.t. $w \in PS^*$, $t, t' \in T_{\text{PSIG}}(\{\mathbf{x}: w\})$ and $\vdash_{\text{ASPEC}}[\mathbf{x}: h(w)]\, h'(t) = h'(t')$. But $[\mathbf{x}: w]t = t'$ holds in every ASPEC-algebra.

Then we can replace 3.10 by

3.13. LEMMA. *If parameter passing is semantically safe then* $f_s(t) \equiv f_{s'}(t')$ *implies* $t \approx t'$ *(w.r.t.* $T_{\text{SPEC}}(A_h)$*) for terms* $t: s$, $t': s' \in T_\Sigma(A_h)$.

Proof. Let A' be the subset of A_h of elements which occur in t or t'. As A' is finite it is of the form $\{\mathbf{a}: w\}$ with $w \in S^*$ and $t, t' \in T_{\text{SIG}}(\{\mathbf{a}: w\})$. By safeness of parameter passing $\vdash_{\text{SPEC}(h)}[\mathbf{a}: w]t = t'$. But then $t \approx t' \bmod T_{\text{SPEC}}(A_h)$ by 3.12. ∎

Then we obtain

3.14. THEOREM. *Parameter is correct for weak parametrized specifications if parameter passing is safe.*

Proof. By minor changes of the proof of 3.4.

Remark. The safeness criteria are satisfied if parameter passing is injective on $B\Sigma$. Though this requirement is rather convenient I must admit that I am not aware of a serious application.

The results in [8] concerning associativity of parameter passing etc. now transfer to order-sorted specifications. Moreover, the proof of Orejas [22] can be applied to establish that, for parametrized specifications, passing compatibility implies parameter protection except for the degenerate case that the formal parameter is inconsistent.

REFERENCES

1. J. A. GOGUEN, J. W. THATCHER, AND E. G. WAGNER, An initial algebra approach to the specification, correctness and implementation of abstract data types, *in* "Current Trends in Programming Methodology," Vol. IV (R. T. Yeh, Ed.) Prentice–Hall, Englewood Cliffs, NJ, 1978.
2. K. BENECKE AND H. REICHEL, Equational partiality, *Algebra Universalis* **16** (1983).
3. R. BURSTALL AND J. GOGUEN, Putting theories together to make specifications, *in* "Proceedings, 5th International Joint Conf. on Artificial Intelligence, 1977."
4. R. BURSTALL AND J. GOGUEN, The semantics of clear, a specification language, *in* "Proceedings, 1979 Copenhagen Winter School of Abstract Software Specification," Lecture Notes in Computer Science, Vol. 86, Springer-Verlag, New York/Berlin, 1980.
5. M. COSTE, Une approche logique des théories definissable par limites projectives finies, manuscript, 1976.
6. H.-D. EHRICH, On the theory of specification, implementation and parametrization of abstract data types, *J. Assoc. Comput. Mach.* **29** (1982).
7. H. EHRIG AND B. MAHR, "Fundamentals of Algebraic Specification 1," Springer-Verlag, New York/Berlin, 1985.
8. H. EHRIG, H.-J. KREOWSKI, J. W. THATCHER, E. G. WAGNER, AND J. B. WRIGHT, Parameter passing in algebraic specification language, *Theoret. Comput. Sci.* **33** (1984).

9. H. EHRIG, W. FEY, F. PARISI-PRESICCE, AND E. K. BLUM, Algebraic theory of module specifications with constraints, *in* "Proceedings, MFCS '86, Lecture Notes in Computer Science, Vol. 239, Springer-Verlag, New York/Berlin, 1986.
10. P. FREYD, Aspects of topoi, *Bull. Austral. Math. Soc.* **7** (1972).
11. K. FUTATSUGI, J. A. GOGUEN, J. JOUANNAUD, AND J. MESEGUER, *in* "Principles of OBJ2," CLSI Rep. 85-22, Stanford University, 1985.
12. P. GABRIEL AND F. ULMER, Lokal präsentierbare Kategorien, Lecture Notes in Math., Vol. 221, Springer-Verlag, New York/Berlin, 1971.
13. M. GOGOLLA, Algebraic Specifications with Subsorts and Declarations, FBNr. 169, Abt. Informatik, Universität Dortmund, 1983; *in* "Proceedings, CAAP '84," Cambridge University Press, Cambridge, 1984.
14. M. GOGOLLA, "Über partiell geordnete Sortenmengen und deren Anwendung zur Fehlerbehandlung in Abstrakten Datentypen," dissertation, Braunschweig, 1986.
15. J. A. GOGUEN, "Order Sorted Algebras," UCLA Comput. Sci. Dept., Semantics and Theory of Comput. Rep. 14, 1978.
16. J. A. GOGUEN, Reusing and interconnecting software components, *Computer* **19** (2), 1986.
17. J. A. GOGUEN, J.-P. JOUANNAUD, AND J. MESEGUER, Operational semantics for order sorted algebras, *in* "ICALP '85," Lecture Notes in Comput. Sci., Vol. 194, Springer-Verlag, New York/Berlin, 1985.
18. J. A. GOGUEN AND J. MESEGUER, Completeness of many-sorted equational logic, *ACM SIGPLAN Notices* **16**, No. 7 (1981).
19. J. A. GOGUEN AND J. MESEGUER, Order-sorted algebra: Partial and overloaded operations, errors and inheritance, SRI International, Computer Science Lab, to appear.
20. S. MCLANE, "Categories," Springer-Verlag, New York/Berlin, 1971.
21. E. G. MANES, "Algebraic Theories," Springer-Verlag, New York/Berlin, 1974.
22. F. OREJAS, Passing compatibility is almost persistency, manuscript, Nancy, 1984.
23. A. POIGNÉ, Another look at parametrization using algebraic specifications with subsorts, *in* "Proceedings MFCS," Lecture Notes in Comput. Sci., Vol. 176, Springer-Verlag, New York/Berlin, 1984.
24. A. POIGNÉ, Error handling for parametrized data types, *in* "Proceedings, 3rd Workshop on Abstract Data Types, Bremen 1984," GI-Fachbericht, Springer-Verlag, New York/Berlin, 1985.
25. A. POIGNÉ, Algebra categorically (Tutorial), *in* "Workshop on Category and Computer Programming, Guildford 1985," Lecture Notes in Comput. Sci., Vol. 240, Springer-Verlag, New York/Berlin, 1986.
26. A. POIGNÉ, Partiality, subsorting and dependant types—Prerequisites of error handling, *in* "Proceedings, Workshop on Abstract Data Types, 1987, Gullane."
27. J. CARTMELL, Generalized algebraic theories and contextual categories, *Ann. Pure Appl. Log.* **32** (1986).